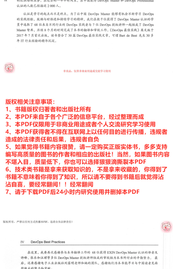
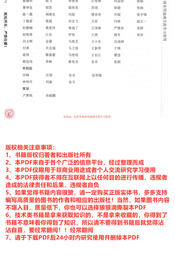




版权相关注意事项：
1、书籍版权归著者和出版社所有
2、本PDF来自于各个广泛的信息平台，经过整理而成
3、本PDF仅限用于非商业用途或者个人交流研究学习使用
4、本PDF获得者不得在互联网上以任何目的的进行传播，违规者造成的法律责任和后果，违规者自负
5、如果觉得书籍内容很赞，请一定购买正版实体书，多多支持编写高质量的图书的作者和相应的出版社！当然，如果图书内容不堪入目，质量低下，你也可以选择狠狠滴撕裂本PDF
6、技术类书籍是拿来获取知识的，不是拿来收藏的，你得到了书籍不意味着你得到了知识，所以请不要得到书籍后就觉得沾沾自喜，要经常翻阅！！经常翻阅
7、请于下载PDF后24小时内研究使用并删掉本PDF

DevOps
Best Practices

DevOps最佳实践

［荷兰］Bart de Best 著　　EXIN DevOps Master智库 译

- ★ DevOps全球资深专家分享30多年实践经验，涵盖规划、编码、构建、测试、发布、部署、运维和监控各个环节。
- ★ 从服务管理的视角阐述DevOps，为企业数字化转型、提升交付产品和服务的敏捷能力提供理念、方法和落地工具。
- ★ 附有EXIN（国际信息科学考试学会）DevOps Master认证备考指南、模拟题及解析。

中国工信出版集团　电子工業出版社 PUBLISHING HOUSE OF ELECTRONICS INDUSTRY http://www.phei.com.cn

内容简介

近年来，许多组织都体会到了使用敏捷方法的好处。敏捷方法使得交付速度更快，质量更高，成本更低。但采用敏捷开发的组织在信息管理、应用管理和基础设施管理方面没有考虑传统的服务管理，这成为采用敏捷开发的一个主要短板。

Dev（开发）Ops（运维）找到了解决这个问题的方法，使开发和运维合并成一个团队，从而共享知识和技能。本书基于作者30多年的IT工作经验，从服务视角阐述DevOps，全面覆盖了规划、编码、构建、测试、发布、部署、运维和监控等组织价值流的各个环节，并把DevOps和服务管理有机结合起来，介绍了DevOps最佳实践，极具指导及参考价值。本书还附有EXIN 的DevOps Master 认证备考指南、模拟题及解析，供希望进一步提升的读者学习。

DevOps
Best Practices

DevOps最佳实践

［荷兰］Bart de Best 著　　EXIN DevOps Master智库 译

電子工業出版社
Publishing House of Electronics Industry
北京·BEIJING

DevOps Best Practices by Bart de Best
ISBN: 978-94-92618-07-8

版权贸易合同登记号　图字：01-2018-2828

图书在版编目（CIP）数据

DevOps 最佳实践 /（荷）巴特·德·贝斯特著；EXIN DevOps Master 智库译. —北京：电子工业出版社，2018.10
书名原文：DevOps Best Practices
ISBN 978-7-121-35116-7

Ⅰ. ①D… Ⅱ. ①巴… ②E… Ⅲ. ①软件工程 Ⅳ. ①TP311.5

中国版本图书馆 CIP 数据核字(2018)第 221599 号

责任编辑：刘露明
印　　刷：三河市鑫金马印装有限公司
装　　订：三河市鑫金马印装有限公司
出版发行：电子工业出版社
　　　　　北京市海淀区万寿路 173 信箱　　邮编 100036
开　　本：787×980　1/16　印张：15　字数：336 千字
版　　次：2018 年 10 月第 1 版
印　　次：2018 年 10 月第 1 次印刷
定　　价：78.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。
质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei. com.cn。
本书咨询联系方式：（010）88254199，sjb@phei.com.cn。

推 荐 序

首先很荣幸有机会为《DevOps 最佳实践》这本书写序。我与该书的作者 Bart de Best 老师相识在 2016 年秋天，时值国际信息科学考试学会（Exam Institute of Information Science，EXIN）刚刚面向全球发布了 DevOps Master 认证体系之际。为了让中国的 DevOps 实践者尽早学习该认证知识体系，我的第一项工作任务是邀请一名既有丰富的 IT 实践经验，又掌握 DevOps Master 认证知识体系的专家来中国开设 DevOps Master 授权讲师研修课程，以此培养出中国本地的 DevOps 授权讲师。在来自全球十几位参与 DevOps Master 认证体系的欧美专家中，我基于行业专业经验和对 EXIN 的贡献这两个基本条件选中了 Bart de Best 先生。

自 1989 年起，Bart de Best 先生担任过 EXIN 早期全球 ITIL Manager 认证考试的阅卷评审工作，30 年之后的今天，他又以行业专家的身份参加了 DevOps Master 认证的全球试题研究工作。更重要的是，我们看重 Bart de Best 先生所具备的 30 多年的 IT 实践和管理经验。他曾在众多世界 500 强公司和政府机构担任过不同岗位的 IT 要职，如联合利华、敦豪速递（DHL）、飞利浦、荷兰皇家航空，以及荷兰银行、ING 银行等众多金融组织。

自 2016 年至今，Bart 老师以主讲师的身份 3 次来到中国培训了几十名中国 DevOps Master 和 DevOps Professional 认证的授权讲师。此后，持有授权讲师资质的中国讲师开展了 DevOps 的认证培训业务。在过去的一年半时间里，在中国获得 DevOps Master 和 DevOps Professional 认证的人数已经接近 2 000 人。

认证是学习的起点而不是终点。为了让中国 DevOps Master 能够有机会不断学习 DevOps 的实践经验，鼓励与时俱进和持续学习的精神，我们在数千位获得了 DevOps Master 认证的学员中选取了 68 位来自不同行业的 DevOps 实践者与 7 位 DevOps 授权讲师一起组成了 DevOps Master 智库，历经 8 个月的时间完成了本书的翻译和审校工作。《DevOps 最佳实践》英文版于 2017 年 7 月首次出版，该书整合了 30 篇 DevOps 最佳实践文章，可谓 Bart de Best 先生 30 多年 IT 行业经验的精华沉淀。

版权相关注意事项：
1、书籍版权归著者和出版社所有
2、本PDF来自于各个广泛的信息平台，经过整理而成
3、本PDF仅限用于非商业用途或者个人交流研究学习使用
4、本PDF获得者不得在互联网上以任何目的的进行传播，违规者造成的法律责任和后果，违规者自负
5、如果觉得书籍内容很赞，请一定购买正版实体书，多多支持编写高质量的图书的作者和相应的出版社！当然，如果图书内容不堪入目，质量低下，你也可以选择狠狠滴撕裂本PDF
6、技术类书籍是拿来获取知识的，不是拿来收藏的，你得到了书籍不意味着你得到了知识，所以请不要得到书籍后就觉得沾沾自喜，要经常翻阅！！经常翻阅
7、请于下载PDF后24小时内研究使用并删掉本PDF

在这里，我要再次感谢参与本书翻译工作的 68 位获得 EXIN DevOps Master 认证的译者及许峰、张乐和汪珺等多位 DevOps Master 授权讲师组成的审校组为本书所付出的辛勤努力。最后，还要感谢电子工业出版社刘露明老师和她的团队，感谢他们为本书能早日与中国读者见面所做出的努力和贡献。

孙振鹏
国际信息科学考试学会 DevOps Days 中国发起人

致中国读者

亲爱的读者，大家好！

知识的分享是我长期以来的动力。我无法想象我的一本或多本著作会被翻译成英文，更不用说中文了。所以，你可以想象《DevOps 最佳实践》一书将被译成中文出版对我来说是一件多么棒的事情了！我为此感到非常荣幸和自豪！

我向参与本书翻译和审校工作的 75 位获得 EXIN DevOps Master 认证者致以诚挚的感谢！我更想感谢的是孙振鹏先生，作为 DevOps 在中国的积极推动者，他发起了中国的 DevOps Days 社区并多次邀请我来到中国担任 EXIN DevOps Master 授权讲师研修课程的主讲人。这让我有机会结识了很多中国的 DevOps 实践者并与他们交流和分享我的实践知识，我从中获得了很大的能力和快乐。对于我而言，这种体验就好比充满了燃料的引擎，给予我在 IT 管理实践领域里继续写作的驱动力。

在我萌生了将我的书翻译成中文出版的想法后，孙振鹏先生将我的书推荐给了电子工业出版社的刘露明老师，《DevOps 最佳实践》才得以作为 DevOps 认证学习的参考教材在中国正式出版。获得认证是学习的起点而不是终点，本书将为获得 EXIN DevOps Master 认证的学员持续学习提供帮助。在孙振鹏先生和 DevOps Days 中国社区的努力下，用了很短的时间组织了 75 位获得 DevOps Master 认证者成立了 EXIN DevOps Master 智库完成了本书的翻译和审校工作。我由衷地希望本书可以帮助更多中国的 DevOps 实践者更好地学习和了解 DevOps。

我也很感谢 EXIN（国际信息科学考试学会），让我有机会参与了 DevOps Master 和 DevOps Professional 国际认证项目的出题工作。要想胜任认证考试出题这项工作，必须在 DevOps 实践领域具备广泛的实践经验和深厚的知识积累。

我也为我的图书出版负责人 Lenard Wolters 先生感到自豪，并对他在本书中文版出版的过程中所付出的努力表示感谢。

我最后要感谢的就是读者你，我真的很感激你对 DevOps 的实践和知识产生了兴趣。当你对本书的内容有任何问题时一定要与我取得联系。

祝你学习愉快！

Bart de Best（bartb@dbmetrics.nl）

致　谢

感谢以下志愿者对本书的翻译及审校做出贡献。

翻译组（按姓氏拼音排序）

敖淳	郭鲲	李平好	申屠欣欣	吴琼	张磊
蔡高扬	郭翔	李锡文	盛永锋	吴婷	张礼军
蔡翔华	侯飞	李智勇	石雪峰	向金珍	张韬
曹坤良	胡罡	梁桂英	苏靖	徐毅	张扬
陈一墨	胡继东	刘晨	屠亚奇	许超	张宇锋
丁晓雷	黄超	刘玄飞	万星	闫林	赵正一
杜静娴	黄星玲	鲁逸丁	王茂森	严惟安	朱婷
封铨贤	江东	罗俊	王涛	姚冬	朱岩
冯文辉	金震	马焕芝	王小东	应成纲	
付勇	开北强	马卫国	王新明	于锋	
高伟	黎嘉豪	马艳琨	王长江	翟进	
顾佳玮	李聃	瞿俊龙	吴非	张娟	

审校组

许峰	汪珺	张乐	张笑梅
林伟丹	刘淼	张引	

策划

卢梦纯　孙振鹏

版权相关注意事项：
1、书籍版权归著者和出版社所有
2、本PDF来自于各个广泛的信息平台，经过整理而成
3、本PDF仅限用于非商业用途或者个人交流研究学习使用
4、本PDF获得者不得在互联网上以任何目的的进行传播，违规者造成的法律责任和后果，违规者自负
5、如果觉得书籍内容很赞，请一定购买正版实体书，多多支持编写高质量的图书的作者和相应的出版社！当然，如果图书内容不堪入目，质量低下，你也可以选择狠狠滴撕裂本PDF
6、技术类书籍是拿来获取知识的，不是拿来收藏的，你得到了书籍不意味着你得到了知识，所以请不要得到书籍后就觉得沾沾自喜，要经常翻阅！！经常翻阅
7、请于下载PDF后24小时内研究使用并删掉本PDF

目　录

1

导 论

1.1 背景

近年来，许多组织都体验到了使用敏捷方法（如 Scrum 和看板）的好处：软件交付更快，质量提高，同时成本降低。但是，不少组织在应用敏捷方法的时候，存在一个主要的缺点，就是没有考虑到传统的服务管理方法，诸如信息管理方法、应用程序管理方法和基础设施管理方法（与敏捷方法的关系）。要避免这个缺点，就应该把目光转向 DevOps 上，就是把开发团队和运维团队合并，形成一个新的管理团队——开发/运维综合团队，我们把它简称为 DevOps 方法，从而让开发管理者和运维管理者都能够共享知识和技能。

1.2 目标

本书以 30 篇最佳实践案例的形式，提供了如何使 DevOps 团队协同工作的知识，并在这些案例中分别提供了关于 DevOps 各环节过程的最佳实践。这些案例涵盖了规划、编码、构建、测试、发布、部署、运维和监控等各个环节（或阶段）。

1.3 目标读者

本书聚焦于 DevOps 团队成员。事实上，正是他们负责服务的全生命周期。本书作者也没有忽略审计人员这一不可或缺的群体。许多最佳实践，都与收集由 DevOps 团队控制的过程的证据（数据）密切相关。

1.4 结构

本书各篇文章的序号、涉及的主题及文章的标题如表 1-1 所示。

表 1-1 文章序号、主题和标题

文章序号	主 题	标 题
#01	基础	DevOps 定义
#02	基础	DevOps 流程
#03	基础	组织模式
#04	体系结构	流程蓝图
#05	体系结构	工具集
#06	体系结构	监控
#07	规划	交付物
#08	规划	瀑布式开发仍会存在
#09	规划	从漏斗到 Scrum 板
#10	规划	服务级别协议和非功能性需求
#11	编码	功能&技术设计
#12	编码	分解特性
#13	编码	定义特性和故事
#14	编码	敏捷变更管理流程
#15	编码	采用静态需求还是动态需求
#16	编码	软件配置项
#17	编码	版本控制
#18	编码	标准、规则和指南
#19	编码	分支模式
#20	编码	异常管理
#21	构建	持续集成
#22	构建	工具
#23	测试	测试类型
#24	测试	测试模式
#25	发布	部署流水线
#26	发布	前向发布

续表

文章序号	主　题	标　题
#27	发布	服务模型
#28	运维	任务划分
#29	监控	持续监控
#30	组织	商业论证
#31	实践	凤凰项目沙盘

1.5 阅读指南

每篇文章一开始，就说明具有专门用法的术语，并说明一个或更多的概念的定义。每篇文章的内容都尽量保持简短，适合快速阅读。

缩略词

在本书中，缩略词的使用是比较有限的，只对规范使用的、前后多次出现的术语使用缩略词，以便使文章更能被轻松地阅读。附录 D 是对本书所使用的所有缩略词的一个总括。

援引

被援引的数字和图表，都用*斜体字*印刷。对文献的援引，都用括号[ ]来表示。附录 A 是参考资料列表。对网站的援引也放在括号[ ]里，附录 E 是参考网站列表。

阅读顺序

本书中的这些文章，每篇都是可以被单独阅读的。对于理解书中的基础知识来说，文章#2 和文章#3 可以说是很好的起点。

术语

在本书中，除非另有说明，术语 ICT（信息通信技术）服务和 ICT 产品分别被定义为“服务”和“产品”。术语“服务”也用来表达交付产品的工作。书中的每篇文章都对该文章中使用的特定术语进行了描述。附录 B 包括了本书使用的所有特定词汇。附录 C 包括了 DevOps 语境中通用的术语。附件 D 列出了缩略词。

2

DevOps 定义（#01）

2.1 DevOps 的起源

近年来，许多组织都体验到了使用敏捷方法（如 Scrum 和看板）的好处。软件交付的速度更快、质量更高的同时成本却更低。然而一个主要的缺点是在信息管理、应用管理和基础设施管理方面，敏捷开发与传统的服务管理相冲突。这主要是因为服务组织无法满足敏捷开发人员的灵活性要求。但一个更重要的原因是，开发和服务管理之间的差距也因此变得越来越大。敏捷所倡导的速度、沟通和协作文化与传统控制型服务管理的组织文化不一致。其结果是，开发团队能够快速交付软件，但软件无法快速发布到生产环境。这一问题需要在合作方式上发生根本性的变化。

2.2 DevOps 是什么

这个问题已经可以通过 DevOps（开发运维）来解决。通过把开发和运维二者合并成一个团队，使知识和技能得以分享，工作方法得以匹配。这对服务管理过程应该如何组织有重大影响。然而，其优势在于既可以频繁地发布，又有高度的控制能力。DevOps 并不像 ITIL 那样有一个明确的统一定义或概念。比如，Gartner 识别了 6 种不同的 DevOps 流程，每种流程对于 DevOps 都有着不同的解释。

2.3 DevOps 的公共特性

虽然 DevOps 没有统一的定义，但是从这些不同的流程中，我们可以找到一些共同的特点，这些特点描述了如何使用 DevOps。

人员

文化、行为、开发、合作、可扩展性和亲和力是 DevOps 的主要内容。

方法

DevOps 的一个重要方面是敏捷化思考和工作，无论是敏捷 Scrum、看板还是其他敏捷形式，而且，精益思想和持续改善也出现在各种 DevOps 出版物中。

资源

资源方面主要强调产品的生产方式，比如版本控制、集成、部署和交付等方面会被经常讨论到，主要侧重在这些方面的自动化。

2.4 DevOps 框架

与 ITIL 等不同，DevOps 并没有被定义成一组最佳实践和流程。尽管如此，一些常用概念的内在关联性仍可被识别出来，如图 2-1 所示。

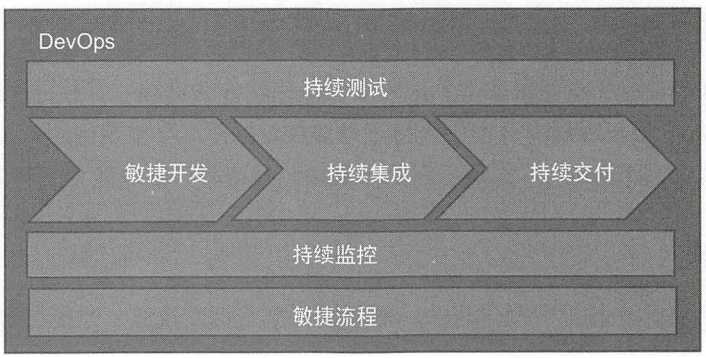


图 2-1 DevOps 框架

这不是一个放之四海而皆准的 DevOps 框架，而只是其中的一个 DevOps 框架。这个框架只指出那些被认可的概念和它们在某种程度上的关系。以下是对每个概念的简要说明。

DevOps 持续测试

持续测试是在整个开发过程中协助测试管理的一个测试方法，包括单元测试、集成测试、

系统测试和验收测试。测试用例最好在软件开发之前编写，而且除了执行常规测试类型外，测试管理也是高度自动化的。要达到这一点，就需要把需求管理、软件配置管理和测试管理高度集成起来。

DevOps 敏捷开发

敏捷开发指的是在 DevOps 中采用敏捷思想进行软件开发，敏捷宣言无疑是很重要的一项。有多种敏捷方法可以采用，比如 Scrum、看板和极限编程。

DevOps 持续集成

持续集成提供了让多个程序员可以同时运行应用程序的最佳实践，可以频繁合并源代码、验证代码（静态测试用例）、编译和测试代码（动态测试用例）。

DevOps 持续交付

持续交付关注从开发、测试、验收到生产环境的高频生产能力。基于高度的自动化，极端的发布上线时间可以达到分钟级。

DevOps 持续监控

持续监控是 DevOps 的重要组成部分，它不仅监控软件（资源），还监控开发人员（人员）和开发过程（方法）。资源在所有环境中被持续地监控，以便尽早发现问题。人员的衡量标准是能力发展（知识、技能和态度），方法层面的衡量则包括速率（处理能力）和效率。

DevOps 敏捷流程

敏捷流程重点关注在标准管理过程中，需要进行哪些调整改进，才能符合敏捷开发方法的要求。

2.5 结论

DevOps 不是一套最佳实践，不是一个模型，也不是一个新的噱头。DevOps 更多的是基于一些基本原则的行动，以便更好地满足客户对上市时间和附加价值方面的需求。其原则包括文化方面、合作方面、通过工具实现自动化和合作的可扩展性。

3

DevOps 流程（#02）

3.1 引言

谈到 DevOps，出现在我们脑海里的是一起工作的一群人，他们不遵循任何框架或 ITIL 这样的最佳实践模型。毕竟每个 DevOps 团队都是独一无二的，他们自己决定如何组织开展自己的工作。这个假设是正确的，但同时，关于这个假设还存在着很多争论。因为这群人所做的工作在本质上是重复性的，而且他们也是有着共同目标的。这也是为什么我们仍然可以在 DevOps 里谈论基于流程驱动的方法的原因。

流程由人员、资源和方法组成。尽管每个 DevOps 团队采用的方法有其独特性，但是流程有着清晰的模式。事实上，法律法规要求这一流程必须受到管控。法律法规是通常意义的，控制也是泛指的。

本文描述了实践中 DevOps 流程的哪些方面构成了一些特定的模式。它也是接下来的一些文章的综述。接下来的每章按照这样的格式——术语、概念、模式、常见问题（FAQ），详述了 DevOps 流程的一个方面。

3.2 流程

图 3-1 显示了一个 DevOps 流程。它不是 DevOps 流程的正式定义，而是表述了在大多数组织机构中，为了实现一个服务而会被循环执行的合乎逻辑顺序的一系列阶段。

深色部分表示开发流程，浅色部分表示运维流程。这两个流程构成了 DevOps 方法的核心。本篇文章不会涉及资源（包括应用程序、基础架构和工具），也不会涉及人员（包括功能、角色和文化），这两部分会在后续文章中进行讨论。

这两部分流程的每一部分又可以进一步细分为一系列阶段、过程，或被称作另一系列流程，它们都是由反复出现的步骤组成的。这些步骤都是为了达到同一个结果，实现相同的目的。

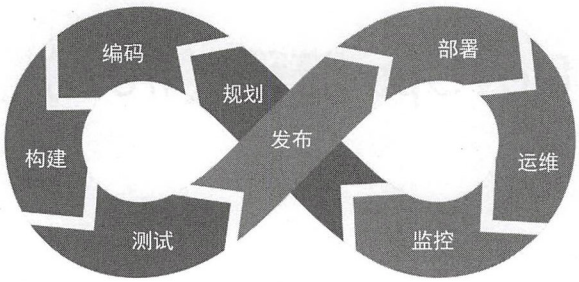


图 3-1 DevOps 流程[CollabNet]

把 DevOps 作为一个流程考虑并就此写一系列文章，并不是为了（也不可能）规范化 DevOps，而是为了用一个更结构化的方式来归类、分享故事及最佳实践。

此外，结构化的 DevOps 将更加容易被定义，从而也更容易被解释。当然，我们需要意识到这些流程虽然是按照顺序画出来的，其实它还包含着一些反馈循环。例如，在测试流程中，为了解决缺陷，有可能触发编码流程。当然，在 DevOps 中，从测试流程触发编码流程是一种浪费，应该尽可能避免。事实上，流程比图上标识的范围更宽泛一些。比如，监控流程也包括了监控整个 DevOps 流程。

最后，每个流程的交付物只包括了最重要的那部分，而不是所有的。接下来的文章会讨论 DevOps 最佳实践。这些文章并不是为了定义 DevOps，而只是一系列的故事，用来阐述这些流程是如何在实践中被应用和实施的。

3.2.1 规划流程

目的

这个流程的目的是为了和相关方一起确定好以下内容：支持客户所需新服务的商业论证（Business Case）、交付的路线图和实现交付的最佳方案。

输出物

- 被任命的相关方
- 确定的业务需求

- 制定的交付路线图
- 选定的交付方案

解释

相关方就是定义功能性和非功能性需求并为之付款的第三方。商业论证说明了为顾客产生的价值及为此所需要的时间、预算和需要控制的风险。交付路线图是一个包含构建模块发布进程的季度时间表。每个构建模块都在史诗（Epic）上定义。方案指的是采用何种开发流程，是敏捷还是瀑布式。在实践中，大多数（80%）采用 DevOps 的组织的商业论证都使用敏捷开发。在这个流程的结束部分，产品待办事项列表已经填好，DevOps 团队已经组建，预算也已经确定。

循着这个最初计划，接下来就会是一个瀑布式项目或 DevOps 方法。在 DevOps 方法中，服务将会采用循环方式增量交付。对每个增量来说，都需要一个详细的计划，史诗被细化为特性，特性被细化为故事。这个方法经常在敏捷 Scrum 开发和看板中使用，但是在实践中任何组合都可能被接受。

3.2.2 编码流程

目的

本流程的目的是产生源代码文件，以便在后续构建流程中，生成需要的应用程序和基础架构组件。

输出物

- 特性被细化为故事
- 确定的功能性或非功能性需求，并且伴随着度量规则
- 定义好的验收测试条件
- 确定好的技术需求
- 定义好的测试用例
- 源文件被保存在一个版本管理系统中

解释

源文件包含了开发人员编写的用来生成应用程序组件的源代码。同时，源文件还包括运维人员创建的用来配置基础架构组件的脚本。为了便于实现，我们使用“特性”来进行描述。特

性是使用业务语言描述的需要交付的功能性或非功能性需求。特性分解后的故事就是需要完成的源代码（应用程序）和脚本（基础架构）的技术描述。

由于经常是多名开发人员在一起编写代码，而代码也会随着时间而改变，因此保持源代码文件和脚本的版本控制非常重要。

3.2.3 构建流程

目的

构建流程的目的是将可应用的源代码转换成目标代码，并执行一系列的单元测试。终极目标是在 5 分钟内验证源代码是否合格，并且将其签入基线版本中。

输出物

- 创建目标代码，并且错误日志中没有错误记录。
- 目标代码被存放在制品库中。
- 源代码已完成质量检查。
- 单元测试执行完成，测试结果无任何错误。
- 源代码被签入基线中。

解释

源代码就是一种适用于某问题域的编程语言，例如，PHP 适用于网站 Web 开发，Java 代码适用于业务功能开发。有些源代码可以直接在生产环境中运行，如 PHP、Basic 和 SQL 编程语言。但是，大多数语言需要把源代码转换成操作系统可读指令（目标代码）。这一过程产物被称为制品。如果源代码扫描和单元测试都是成功的，就把源代码签入基线中。

单元测试不同于其他类型的测试，因为这些测试用例保留了目标代码的代码环境。鉴于 5 分钟构建的要求，因此不需要再对目标代码中需与其他应用组件或基础架构的协作进行测试检查。在 DevOps 团队中共享的源代码一起形成一个新的服务。

3.2.4 测试流程

目的

测试流程的目的是在当前冲刺中生成的所有源代码和基础架构脚本上，执行集成测试和系

统测试。

输出物

- 系统和集成测试被成功执行。
- 服务符合技术要求和度量要求。
- 技术验收测试执行完成，测试结果无任何错误。

解释

集成测试被定义为测试应用程序组件间的协作。系统测试被定义为测试端到端的服务实现，包括对基础架构的测试。

3.2.5 发布流程

目的

发布流程的目的是判定服务的新版本是否满足其功能性及非功能性需求。

输出物

- 服务符合规定的功能性及非功能性需求和度量要求。
- 验收测试执行完成，测试结果无任何错误。

解释

在 DevOps 中，发布流程和部署流程有所区别。发布流程是关于新版本服务的正确性的验证，部署流程是为分发服务的新版本而服务的。新版本服务的发布则是基于相关需求的测试结果。

3.2.6 部署流程

目的

部署流程的目的是通过部署流水线尽可能实现以自动化的方式在生产环境中推出新服务。

输出物

基础架构脚本化的生产环境。

解释

自动化创建服务新版本是 DevOps 的一个重要方面，其实现机制就是部署流水线。实际上部署流水线并不只局限于部署过程。部署流水线从设定需求就已开始，其格式可度量并可以自动化（编码过程），最终自动化发布新服务和自动化安装补丁（运维过程），以保持服务永远满足需求。另外，旧版本服务的重置回滚也是部署流水线的功能的一部分。

3.2.7 运维流程

目的

运维流程的目的是保持服务的稳定可靠，并且运行符合约定的功能和质量要求，保证业务的连续性。

输出物

- 安全的环境和数据。
- 执行管理任务。
- 预防事件发生。
- 观察并记录发生的事件。

解释

保持服务正常运行实际上包含多个流程，包括最底层的基础架构（基础架构管理）、应用程序（应用程序管理）和数据（信息管理）。为了使服务正常工作，在编码流程需要采取多种测量。请注意，应对程序 Bug 的解决方案并不是本流程的一部分，而属于编码流程。

3.2.8 监控流程

目的

监控流程的目的是监控约定的服务标准。其范围不仅仅包括生产环境，还涉及整个 DevOps 流程。

输出物

- 已建立的监控指南。
- 已定义的事件目录。
- 已装配的监控设施。
- 确定（潜在）的生产标准偏差。
- 已登记的事件。
- 服务报告。

解释

监控合适的运维操作实际上包含多个流程，包括底层的基础架构、应用程序和数据。为了使服务正常工作，我们需要在编码流程中采取多种测量。请注意，列入事件（inclusion of incidents）并不是监控过程的一部分，而是编码流程的一部分。

3.2.9 DevOps 框架结构关系

图 3-2 给出了 DevOps 流程与 DevOps 框架间的关系概览。

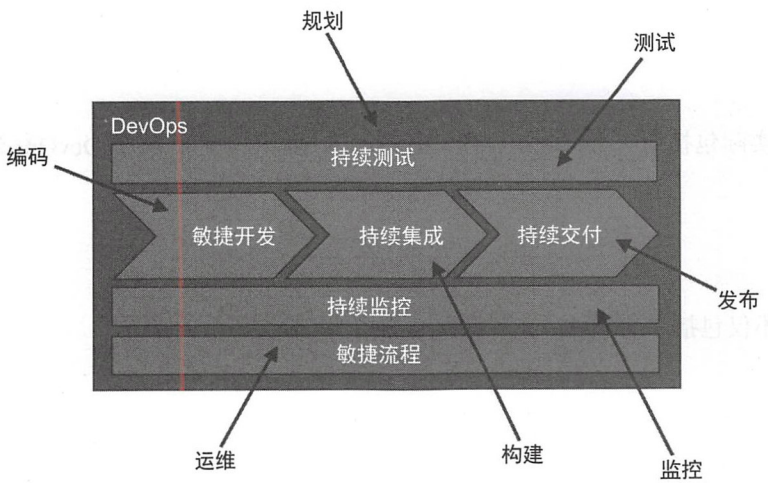


图 3-2 DevOps 流程与 DevOps 框架间的关系

图 3-2 是指示性的。图中没有画出清晰的线条，但它向我们展示了其连贯性。以下是对图 3-2 的简要解释，需要说明的是这种关系不是纯粹的一对一的关系。

规划

规划包含所有 DevOps 活动，既包含最初的整个路线图，又包含服务最后的增量交付。

编码

敏捷开发主要涉及编码流程中提及的各个方面。

构建

持续集成主要包括构建流程，也包含单元测试。

测试

持续测试在本文中比测试流程范围更大，因为它包括全生命周期中所有测试类型，如构建流程中的单元测试用例。

发布

持续交付不仅是一次发布的推出，还包括部署流水线，这已经在敏捷开发可执行的测试用例中被定义了。

运维

敏捷流程实际包括所有 DevOps 流程，而不仅仅是运维流程。整个 DevOps 流程就是敏捷流程。

监控

持续监控不仅包括产品阶段，还包括整个 DevOps 流程。

4

组织模式（#03）

4.1 引言

本文描述了几种 DevOps 团队的组织模式。

4.2 术语

服务台

服务台是客户和服务供应商之间的单一接触点（SPOC）。二者之间主要的沟通内容包括问题、服务请求、事件、产品特性需求和投诉。服务台对于 DevOps 团队尤为重要，因为它使得 DevOps 团队可以专注于其流程。

变更控制

在每个 DevOps 团队内都会通过一种关卡的形式来接受一个产品特性并推进它。这种控制被归纳为一个术语“变更控制”。它可以有多种组织形式，如产品负责人、把关人、变更顾问委员会（CAB）。根据产品特性的风险和影响，同一个 DevOps 团队可以选择不同的变更控制。

垂直特性分割

在理想的情况下，一个 DevOps 团队能够独立自主地实现一个产品特性。这意味着他们的工作不需要依赖其他 DevOps 团队或其他组织单位，如基础架构部门。这种方式的优点是使得团队之间的沟通和协作最小化，缺点是这个团队必须有丰富的知识和专业技术。

水平特性分割

在一些服务组织中，需要一个专门的 DevOps 团队，他们只负责实现某个产品特性需求的一部分。在这种情况下，一个产品特性需求需要更多的 DevOps 团队一起协作才能最终发布这一产品特性。其优点是这些团队对其负责的产品模块有深度的知识并能够交付高质量产品，缺点是几乎每个产品特性都需要两个以上团队协作才能完成系统上线。

模式

模式是针对某个通用问题或反复出现的问题的通用解决方案。一个 DevOps 团队可以通过分析其遇到的问题及解决问题的方法，总结出模式，并通过叙事分享给其他 DevOps 团队。本文给出了一些关于如何组建 DevOps 团队的模式。

4.3 概念

特性分割

在 DevOps 团队中有两种方法来细分产品特性，即垂直特性分割和水平特性分割。垂直特性分割意味着一个 DevOps 团队要处理整个产品特性。水平特性分割意味着需要多个 DevOps 团队共同来实现一个产品特性。

一般来说，我们首选垂直分割，因为这只需要较少的团队间交互。然而，这并不是总能奏效的，比如当应用程序与/或基础架构过于复杂时。本文将会提及每个模式所使用的特性分割方法。

4.4 模式

在组建 DevOps 团队时已经识别出了多种模式，每种模式都有其优点和缺点。本文主要讨论以下 3 种模式：

- 业务线模式。在这种模式下 DevOps 团队只为一条业务线工作。
- 组合模式。这种模式基于应用程序组合中的各个应用来划分 DevOps 团队。
- 信息价值链模式。这个模式特别适用于信息提供商把信息池（information lake）转化为报告。

这 3 种模式都应用了服务台和变更控制的各种职能，但每种模式对垂直分割和水平分割的使用方法是不同的。采用的模式不同，对 DevOps 团队的绩效及其工作质量会产生很大影响。

4.4.1 业务线模式

业务线模式是按照应用程序来划分 DevOps 团队的，所有的应用程序根据不同的业务线来分割，在各个 DevOps 团队中的交互为零（见图 4-1）。因此这是一种垂直特性分割。

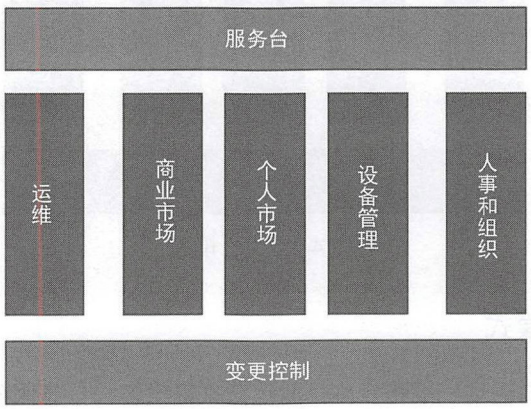


图 4-1 业务线模式

在这种模式下，服务台是客户和服务供应商之间的单一接触点。运维团队负责监控服务。变更控制是一个控制单元，在不同的 DevOps 团队间规划业务特性请求及系统上线。这个变更控制是轻量级的，因为不同 DevOps 团队之间负责的产品特性很少被共享。

这些业务线是：

- 目标为业务的商业市场。
- 目标为自然人的个人市场。
- 在器材、停车场、餐馆及其他服务领域中提供服务的设备公司。

4.4.2 组合模式

在这种组织形式中，团队是按照应用程序的功能而聚集的。同样，这种模式也是基于垂直特性分割的。

这些通用的 DevOps 团队（服务台、运维、变更控制）与业务线模式相同。应用程序集群的例子是运维、财务应用、资源管理应用、人力资源管理应用及人事和组织应用（见图 4-2）。

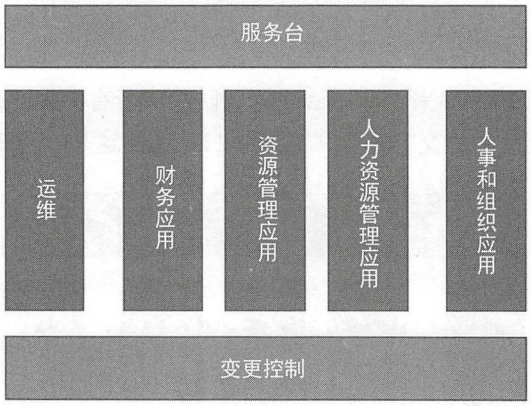


图 4-2 组合模式

4.4.3 信息价值链模式

信息价值链模式试图通过一组相互协作的团队来提供客户信息服务（见图 4-3）。这就意味着 DevOps 报告团队想要设计一个新的报告，需要 1～5 个 DevOps 团队参与。因此，这是水平特性分割（参见文章#12）。

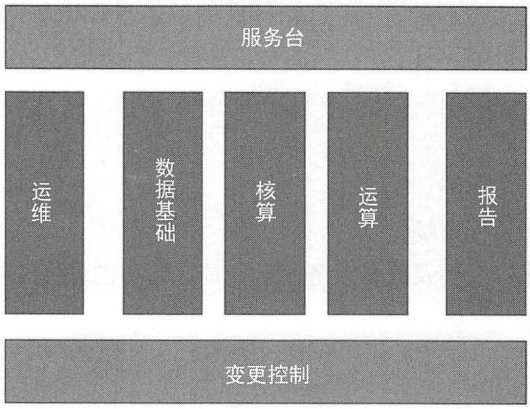


图 4-3 信息价值链模式

在这种模式中，服务台是客户和服务供应商之间的单一接触点。运维团队负责监控信息服务。数据基础团队负责收集信息文件。核算团队负责管理信息文件的处理流程及质量审计。运

算团队负责执行报告中需要的数据计算。报告团队负责将计算好的数据提供给客户，报告形式包括数据市场、报告模板及应急报告。变更控制是一个控制单元，在不同的 DevOps 团队间规划业务特性请求及系统上线。

选择这种模式的原因是，如创建报告这种情况下，所需要用到的知识和专门技术太广泛。也可以改变这种模式，根据报告内容来为 DevOps 团队分组。然而，这就需要 DevOps 团队必须能够完成数据收集、数据核算、数据计算和数据报告的全部任务。因此，当各个团队工作于相同的应用程序和信息对象时，团队之间就会相互影响。

4.5 常见问题（见表 4-1）

表 4-1 组织模式的常见问题

序号	分类	问 题	答 案
1	选择	有更多的模式吗？	肯定会有更多的模式。一旦它们被识别，将会被添加到本文中
2	延迟	一个外部变更控制会使 DevOps 团队速度变慢吗？	是的。无论什么情况下，如果可能，都应该在严格需要时才引入控制

5

流程蓝图（#04）

5.1 引言

本文描述了一种体系结构模型（architecture model），用它在以 DevOps 为基础的服务组织中实现服务管理流程的平衡设计。

5.2 术语

体系结构模型

体系结构模型是对现实情况的简化呈现。

5.3 概念

流程蓝图

流程蓝图是体系结构模型的一个例子。蓝图用一张图显示了在服务组织中使用哪些服务管理流程，以及在没有详细显示所有沟通路径的情况下，如何组织纵向和横向治理。

表 5-1 展示了一个 16 区域的服务管理模型。这个模型是通过 DevOps 流程完成的，对于每个组织来说都是非常有启发性的。如果流程没有被明确识别，那么也可以用可交付成果来完善这个 16 区域模型。为了简单明了起见，本文只阐明可交付成果。另外，DevOps 团队当然也可以在单元格中加入更多内容。

表 5-1 16 区域服务管理模型

指导层面	用户组织	信息管理	应用管理	基础架构管理
战略指导				
战术指导				
运维指导				
创新指导				

这种管理模型的优势在于，它可以帮助您思考 DevOps 领域内有什么和没有什么。一旦有些内容被排除在 DevOps 领域之外，那么，可交付成果在哪里被覆盖就成了问题。

5.4 模式

本文讨论了 16 区域服务管理模型的一些典型的实现方法。这些陈述是象征性的而不是限制性的，因此，对它们可能还会有许多额外的补充。这些示例的目的是阐明在 DevOps 团队中实施何种“控制”是如何选择的。另外，重要的是，要通过识别和承担结果来明确地把握具体的控制方面。

5.4.1 传统服务组织模式

在传统的服务组织中，管理模型的每个单元格通常都会被填入内容。在通常情况下，使用的是 BiSL 和 ITIL 流程的可交付成果。表 5-2 显示了一个示例。

表 5-2 典型服务组织的 16 区域模型

指导层面	用户组织	信息管理	应用管理	基础架构管理
战略指导	• 使命、愿景和战略 • 业务目标 • 信息规划	• 信息组合 • 体系结构场景 • 信息路线图	• 应用组合 • 体系结构场景 • 应用路线图	• 基础架构组合 • 体系结构场景 • 基础架构路线图
战术指导	• SLA 评审	• 年度和季度信息管理 • 商业论证 • 资源计划 • 财务计划	• 应用发布计划 • SLA • 安全计划 • 持续计划 • 可用性计划	• 基础设施发布计划 • SLA • 安全计划 • 持续性计划 • 可用性计划

续表

指导层面	用户组织	信息管理	应用管理	基础架构管理
战术指导			• 容量计划 • 资源计划 • 合同	• 容量计划 • 基础架构规划和设计 • 合同
运维指导	• 超级用户 • 升级 • 呼叫	信息事件	• 应用事件 • 事件解决方案 • 根本原因分析 • 事态管理 • 性能和调优	• 基础设施事件 • 事件解决方案 • 根本原因分析 • 工作量调度 • 监控 • 性能和调优 • 环境管理
创新指导	请求变更	• 风险评估 • 请求变更日历 • 需求 • UAT，FAT	• 风险评估 • 应用代码 • 基线 • UT，SIT，PAT	• 风险评估 • 脚本 • 基线 • UT，SIT，PAT

5.4.2 敏捷开发团队模式

对荷兰 10 个组织的调研[Best 2015a]表明，在使用敏捷方法的服务组织中，一直在使用服务管理流程。表 5-3 显示了这 10 个组织分配给敏捷开发团队的工作范围。单元格中的这些数字反映了这 10 个组织中实践敏捷方法的数量。不同的颜色有不同的意思。其中对信息管理的忽视很令人担忧，从而也可以看出，它们在战略框架的大多数方面的缺乏是十分惊人的。

表 5-3 敏捷开发团队 16 区域模型[Best 2015a]

指导层面	用户组织	信息管理	应用管理	基础架构管理
战略指导	—	1	3	1
战术指导	—	4	5	1
运维指导	—	2	7	—
创新指导	—	5	10	1

□ 不涉及（灰色）
■ 几乎不涉及（红色）
■ 部分涉及（黄色）
■ 涉及（绿色）

表 5-4 显示的是可能被组织所期望的可交付成果的示例[Best 2015a]。对于这些组织来说，一个重要问题是：对管理架构的指令（服务战略）和规划（服务设计）的其他单元而言，什么是必须被实现的。

表 5-4 敏捷开发团队 16 区域模型

指导层面	用户组织	信息管理	应用管理	基础架构管理
战略指导	×	×	×	×
战术指导	×	×	产品待办事项列表	×
运维指导	×	×	应用事件	×
创新指导	×	需求，UAT, FAT	应用代码，基线，UT, SIT, PAT	×

5.4.3 DevOps 团队的模式

除了传统管理组织模式和敏捷开发团队模式以外，我们更乐于看到 DevOps 组织的期望。在表 5-5 中，一个组织采用了 DevOps 方法。在这种情况下，它是一个业务 DevOps 团队，因为信息管理在其范围内。这种实现是基于 DevOps 团队本身的实际工作，而且是被线性组织覆盖的。

表 5-5 详细的 DevOps 团队开发 16 区域模型

指导层面	用户组织	信息管理	应用管理	基础架构管理
战略指导	• *使命、愿景和战略* • *商业目标* • *信息规划*	• *愿景声明* • *信息组合* • *架构场景* • *信息路线图*	• *应用组合* • *架构场景* • *应用路线图*	• *基础架构组合* • *架构场景* • *基础架构路线图*
战术指导	SLA 评审	• *年度和季度信息管理规划* • *商业论证* • *资源计划*	• *应用发布计划* • *SLA* • *持续性计划* • *安全性计划* • *可用性计划* • *容量计划* • *资源计划* • *合同* • 产品待办事项列表	• *基础架构发布计划* • *SLA* • *持续性计划* • *安全性计划* • *可用性计划* • *容量计划* • *基础架构规划和设计* • *合同*
运维指导	• 超级用户 • 升级 • 呼叫	• 信息事件 • 看板	• 应用事件 • 事项解决方案 • 根本原因分析	• 基础架构事件 • 事件解决方案 • 根本原因分析

续表

指导层面	用户组织	信息管理	应用管理	基础架构管理
运维指导			• 事件管理 • 性能调优	• 看板 • *机房管理** • *工作负载* • *调度* • *监控* • *性能调优* • *环境管理*
创新指导	• 产品待办事项列表 • 特性需求列表	• *风险评估* • 特性定义 • *需求* • GAT，FAT • *Scrum 看板*	• *风险评估* • 应用代码，基线 • UT, SIT, PAT • Scrum 看板	• *风险评估* • 脚本，基线 • UT, SIT, PAT • Scrum 看板

* 机房管理（House-keeping）放在基础设施的那一列，所以应该是相关房间的管理及一些日常巡检之类的工作。

在表 5-5 中，没有被使用的可交付成果用斜体标出。在这里，涉及的是架构方面、规划方面和控制方面的可交付成果。用正体标出的与 DevOps 直接相关。这个结果对起步阶段的 DevOps 组织来说是非常普遍的。部分原因是一个 DevOps 团队常常是从开发和维护一个网站开始的，因此，似乎不太需要指令（架构）和控制（规划）。随后通常会出现后台需要的一些接口，一旦这些被开发出来，在特定的压力下，会出现需要更多的控制和指令的情况。

5.5 常见问题

表 5-6 中包含了与流程蓝图相关的常见问题。

表 5-6 流程蓝图的常见问题

序号	主　题	问　题	解　答
1	覆盖范围	理想的 DevOps 范围是什么？	DevOps 理应涵盖一个服务的全生命周期，无论是在信息、应用还是在基础架构管理等方面。在任何情况下，运维层面和创新层面都是 DevOps 团队的工作主题。对于战术和战略流程，通常会选择由多个 DevOps 团队集中来实现。无论如何，指令

续表

序号	主　题	问　题	解　答
1	覆盖范围		（战略层面）和规划（战术层面）工作要和 DevOps 团队密切合作，这很重要。安全架构和 SLA 标准就是一个例子
2	基础架构范畴	DevOps 团队通常拥有基础架构管理的全部控制权吗？	很多组织将基础架构管理划分到一个通用层面上，并且将 DevOps 团队都设置在该层之上。这些 DevOps 团队可以在一定的限制范围内自行对基础架构进行编程

6

工具集（#05）

6.1 引言

本文描述了应用、基础设施架构如何促成 DevOps 工具的选择。

6.2 术语

工具集

工具集是工具使用情况的概览。基于工具集，我们可以分析工具缺失、冗余、多重功能等场景。使用工具集的一个重要出发点，就是防止工具间低效协作引发 DevOps 团队功能缺失、质量蜕化。

开发或购买

维护工具集，涉及确认是自己开发还是购买一款工具。这并非一个非黑即白的选择，往往要在深度定制已购工具，或者采购更贵、有更多可用特性的工具间做出选择。

最佳组合/工具集成

选择工具的关键要素是功能（工具提供最佳功能或相近功能），或者关注由单个或多个工具联合覆盖全部功能需求。

6.3 概念

生命周期管理

工具如人，具有生命周期。Gartner 技术成熟度曲线是分析工具位于生命周期何处的重要工具。因为新工具具备新技术、新洞见，启用市场上全新的工具往往能带来优势。但全新工具也意味着市场接受度还比较低，这意味着可能不被用户采纳，或者会从市场上消失。这样的新工具往往缺乏稳定性，日渐增多的痛点有待消除。

6.4 最佳实践

6.4.1 所有权

工具集必须有所有者。通常而言，每个 DevOps 团队都想声称自己是应用工具的所有者，因为团队必须承担这样的角色，也希望拥有自主能力。尽管开发人员或系统管理员都对某种工具存在偏好，但组织中的团队常常协同工作，如 DevOps 团队工作中存在一个链或项目需要长期运维。在这种情况下，DevOps 团队使用相同的工具变得较为重要。

因此，在工具集中分层变得很重要。第一层是通用工具，如服务台工具、部署工具或调度工具。该层工具集通常由架构（师）负责。这意味着所有团队都需要使用它们。第二层由一系列特定功能的工具组成，通常这些工具仅仅实现一部分特定功能，使用者需要做出选择。例如，一个团队会选择专用于 Java 应用的监控工具，而别的团队为了更好地度量网络情况选择了另一个。基础架构必须保证这些工具配合良好，诸如将日志写入集中事态管理（central events manager）。最后，第三层由员工自主选择，作为工具集的空位补缺。在整体架构中，为改进、创新预留空间也很必要。

6.4.2 工具集框架

观察工具的应用情况，对决定一个工具集的完备性很重要。对于 DevOps 架构，你可以参照 DevOps 过程逐步核查（参见文章#02）。按过程中每步决定的所需功能，经测量去选择可用工具。表 6-1 给出了 10 个组织已识别出的功能。

表 6-1 DevOps 工具功能

DevOps 过程	功 能
规划	特性管理 产品代办清单 迭代代办清单 仪表盘 变更管理
编码	软件配置管理 代码库 版本管理 基线
构建	持续集成
测试	缺陷跟踪 测试用例 测试脚本 回归测试
发布	安全管理
部署	发布部署管理 持续交付 开发—测试—验收—生产（DTAP）
运营	事件管理 问题管理 配置管理
监控	报告

6.4.3 工具集分析

图 6-1 给出了 10 个组织使用的工具的总览图，工具的数量着实可观，而且许多工具不止一个功能。

工具	特性/用户故事管理	冲刺待办事项列表	仪表盘	软件配置管理	代码仓库	代码评审	版本/基线管理	持续集成	问题跟踪	测试用例/测试脚本	测试自动化	回归测试	发生管理	事件管理	问题管理	变更管理	配置管理	发布及部署管理	持续交付	开发—测试—验收—生产	报告	合计
	敏捷项目管理							测试管理					服务管理									
#define	1			1		1			1	1			1							1		7
Bamboo								1														1
CCCQ				1	1	1	1							1	1		1					7
Cherwell													1	1	1							3
ClearCase				2	1	1	1										1					6
ClearQuest							1		1					1	1		1					5
Clientele									1				2	1		1	1	1				7
Enterprise Architect	2	1																1				4
FitNesse											1											1
GIT				1	2		2		1													6
HP ALM	1								2	3	2	3			1		1					13
HP QC																						0
HP SC				1									1	1	1	1		1				6
HP Service Manager													1	1	1	1						4
IBM Tivoli Service Management Suite													1	1								2
Ice-Scrum	1																					1
Installshield																	1					1
Jenkins								5												1	1	7
Jira	3	4	3				1		2						1							14
JMeter											2	4										6
Leankit	2	1	1										1		1							6
Maven				1				2			1									1	1	6
Mavim								1														1
MCC																						0
MRM																						0
McAfee																			1			1
MScc																						0
Mediawiki															1							1
MS Excel										1							1	1				3
MSI				1	1	1	1	1							1	1	1			1	1	10
MTM										1	1	1										3
Nexus				1				1														2
NPS			1																			1
Octopus Deploy								1														1
Omnitracker													1	1	1		1					4
OTRS									1				1									2
Overig																			1			1
Powershell																				1	1	2
ReportServer			1															1				2
Robot Framework												1										1
ROSS																	1			1	1	3
ScrumWise	1	1	1																			3
Selenium										1	4	4										9
SonarSource																						0
Serena Dimensions					1																	1
SOAtest										2	1	2										5
Sonar						2		2														4
Subversion					2	2	3															7
TestComplete										1	1	1										3
winMerge																						0
TFS	2	2	2	1	2	2	3	1	1	1	1	1			1		1	1		1	1	24
TOPdesk													1	1	2	1	2	1				8
Twist											1	1										2
合计	13	9	9	10	10	10	13	15	10	11	15	18	11	9	14	5	13	7	2	7	6	

图 6-1 工具集分析

6.4.4 陷阱

筒仓选择（Silo Selection）

选择每个种类里的最佳工具。对于 DevOps 过程尚不成熟的团队，工作往往是片段式的而非集成式的，这样的选择也就很常见了。然而，当过程越来越成熟，每阶段各步骤会涉及越来越多的信息传递，自动化的需求同步增长，此时，往往工具间无法轻易贯通，甚至根本无法连接。因此，在市场中有一种使用集成解决方案的趋势，以期问题不再出现。此类集成工具的缺点在于单一领域延展性功能较少。通过观察单个工具提供了多少特性，在图 6-1 中比较容易识别出哪些工具提供集成解决方案。为了做出好的选择，应关注各环节中对过程重要的功能。与此同时，依然会需要额外的工具或连接来弥补空缺。

自下而上

当 DevOps 团队每每遭遇问题进而选择工具时，常会落入巨大的陷阱。很多场景下，DevOps 团队需要交流信息，却由于成本过高看似不可能实现。此时，往往标准化会成为更好的选择。研发人员和系统管理员像选择泰迪熊似地挑选工具，个性十足又钟爱有加。大量的时间被花在工具升级上（指工具被定制化二次开发。——译者注），导致他们感觉自己成为工具的所有者。其后果是这些工具无法被新的、有更标准的功能的工具替代。尽管市面上有更好的、更标准的可行解决方案，他们还是会持续开发和连接这些工具。

自制或采购

没有多少服务组织自行开发工具，但定制化的工具和插件却很常见。

根据定义，总有需要配置的东西，也总有实现工具之间连接的要求。问题是要消耗多少时间和精力，市面上是否存在既有集成解决方案。此外，由于仅有一位专门的人能够保持工具运转，这会产生员工锁定（employee lock）现象。

7

监控（#06）

7.1 引言

本文描述了应用程序和基础设施架构如何影响 DevOps 监控工具的选择和使用。

7.2 术语

监控分层模型

有成千上万的监控工具，从中可以选择用于部署流水线和监控生产的工具。

事态黑名单

监控工具触发通知的事态会被列入黑名单，这意味着当这些事态发生时，将会触发告警，运维团队随后便能采取相应措施。

事态白名单

无关的事态会被列入白名单。

误报

应用程序引发异常的错误，而实际上该错误未发生。

漏报

应用程序没有在事态中触发错误，而实际上错误的情况已经发生。

7.3 概念

监控分层模型

有成千上万的监控工具，从中可以选择用于部署流水线和监控生产的工具。

图 7-1 给出了可用于监控服务的各种监控功能概览。

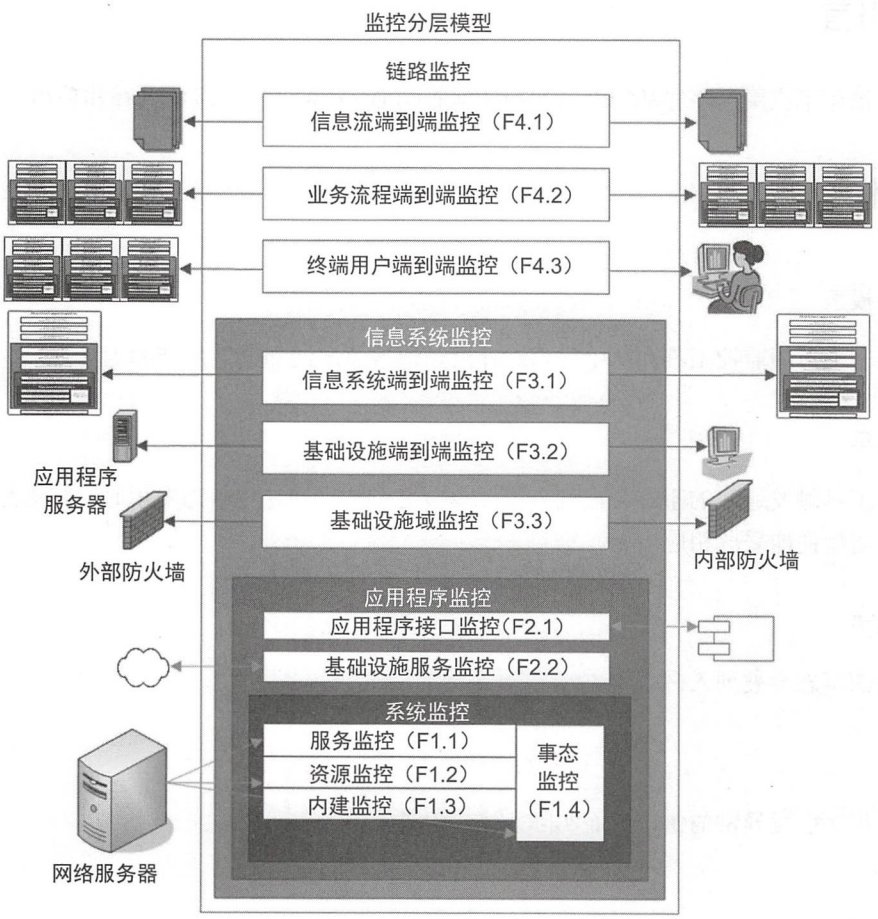


图 7-1 监控分层模型[BEST 2015b]

F1. 系统监控（单元监控）

系统监控是监控分层模型的基础，对所有应用程序和基础设施单元进行度量。系统监控由于其单元层级的特性而有别于其他监控形式，因此系统监控也被称作单元监控。只有个别应用程序和基础设施组件不需要通过它来度量。系统监控对于基础设施的技术管理至关重要，系统监控由以下监控功能组成（见表 7-1）。

表 7-1 系统监控

序号	分 类	描 述
F1.1	服务监控	同时监控基础设施服务和应用程序服务。这些服务通过一种管理协议来度量，通常使用简单网络管理协议（SNMP）
F1.2	资源监控	监控某一单独组件的资源使用状况，如 CPU、内存和网络带宽
F1.3	内建监控	内建监控是写在应用程序里的监控机制，它的价值在于可以度量应用程序运行态（application runtime）记录，而这在应用程序之外是不可见的
F1.4	事态监控	事态监控（Event Monitoring）包含了所有的基础设施组件和基础设施服务，以及应用程序事态。通常通过读取本地日志来获知事态，如应用程序日志、安全日志和错误日志

F2. 应用程序监控

通过度量底层系统（系统监控）、基础设施服务和应用程序接口，应用程序监控为一个独立应用程序（子系统）或服务提供信息全景图。这一层对于应用程序管理而言很重要，它在一个视图中显示出来自一个应用程序的所有相关信息。应用程序监控包含系统监控功能的同时，还具备如下监控功能（见表 7-2）。

表 7-2 应用程序监控

序号	分 类	描 述
F2.1	应用程序接口监控	应用程序接口监控是特别为应用程序管理员们准备的，这样他们可以一目了然地确定应用程序是否正在按照预期运行。因此，很有必要的是，每个应用程序都需要明确哪些基础设施组件、基础设施服务和应用程序服务是重要的。另外，应用程序层会建立度量，检验基础设施和应用程序代码之间的交互

续表

序号	分　类	描　述
F2.2	基础设施服务监控	虽然一个基础设施服务可能是“在线”，但并不意味着它在正常工作或可访问。为了确定服务是否正常工作，需要在服务被调用的基础设施层运行动态测试

F3. 信息系统监控

需要多个应用程序实现一个用户功能，但这些应用程序往往不被用户单独识别。为了在信息系统层制定用户协议，在应用程序监控层之上，我们定义了信息系统监测层，本质上它是一系列的应用程序。度量一个端到端（E2E）的信息系统，应尽可能从用户层面来开展监控，这些度量通常通过遍历业务路径来实现，因此这一级的监控又被称作业务监控。

E2E 基础设施度量也被用于支持这一度量，E2E 基础设施监控展示了基础设施各种组件间是否存在关联，在地理位置分离和具有更多逻辑网络（局域网 LAN、广域网 WAN 等）的大型计算中心尤其如此。信息系统监控在以下功能上作为应用程序监控的补充（见表 7-3）。

表 7-3 信息系统监控

序号	分　类	描　述
F3.1	信息系统 E2E 监控	单个应用程序可用并不意味着用户可以正常使用，通常多个应用程序结合在一起才能完成整个链路（信息系统）
F3.2	基础设施 E2E 监控	E2E 监控基础设施包含整个基础设施层的度量，这种监控通常应用程序于广域网（WAN）
F3.3	基础设施域监控	在具有多个供应商或网络被分割成单独的物理/逻辑域场景下，从可用性等考虑，需要单独监控这些域。相较于自动化办公网络，诸如隔离域（DMZ）可能采取其他标准

F4. 链路监控

链路监控旨在从不同的视角反映整个链路的运行情况：

- 用户感知。
- 业务流程所需的性能。
- 链路中的信息处理。

除了信息系统监控外，链路监控还包括以下功能（见表 7-4）。

表 7-4 链路监控

序号	分 类	描 述
F4.1	信息流 E2E 监控	在这个水平上，识别了信息流监控。E2E 的信息流度量方法，不是度量信息和通信技术服务，而是度量纯粹的业务处理信息流
F4.2	业务流程 E2E 监控	基于相关的信息系统，这个功能度量了从前端到后台的业务流程
F4.3	终端用户 E2E 监控	在这个水平上，度量用户接受的服务。从最终用户的视角，展示提供的服务

7.4 最佳实践

DevOps 流程包括监控阶段。监控服务是 DevOps 的基础。监控功能旨在监控服务的正常运行，提前发现一些事件。另外，监控服务能度量 SLA 标准。但是实现一个健全的监控服务并不是一件易事。为了提供健全的监控能力，这篇文章会介绍 DevOps 流程中需要考虑什么。

7.4.1 监控原则

以下是一些适用于确保监控正确使用的原则（见表 7-5）。

表 7-5 监控原则

分 类	描 述
监控功能	监控功能必须和 SLA 一致
相关性	除了 E2E 监控，也必须实现系统监控
监控分类	对于要监控的每类对象，尽可能使用相同的监控工具

7.4.2 监控工具和 SLA

生产环境中监控工具的选择依赖 SLA 的内容。举个例子，如果 SLA 仅明确阐述了硬件平台和网络组件层面的运行指标和标准，那么对于 SLA 标准的监控，使用系统监控工具就足够了。但是如果 SLA 描述了信息系统层面的运行指标，那么就需要调整监控，以适应其要求。

有些工具是将一个信息系统看作一组潜在对象的集合，将可用性、容量等定义为集合中的一个功能。其他工具则监控信息系统本身，如监控交易流。监控工具的功能只有和 SLA 标准的层级相匹配，才能提供一份可靠的 SLA 报告。

7.4.3 DevOps 流程

表 7-6 提供了与每个 DevOps 阶段及监控功能相关的 DevOps 监控综览。之所以省略了“运维”阶段，是因为该阶段没有需要填写的内容。

表 7-6 每个 DevOps 阶段的监控功能

序号	监控功能	计 划	编 码	构 建	测 试	发布	监 控
F1.1	服务监控	工具选择	—	—	系统测试	部署	异常响应
F1.2	资源监控	工具选择	—	—	系统测试	部署	异常响应
F1.3	内建监控	—	开发监控	单元测试	系统测试	部署	异常响应
F1.4	事态监控	工具选择	开发异常	单元测试	系统测试	部署	异常响应
F2.1	应用程序接口监控	—	开发异常	—	集成测试	部署	异常响应
F2.2	基础设施服务监控	工具选择	—	—	系统测试	部署	异常响应
F3.1	信息系统 E2E 监控	工具监控	开发伪交易	—	终端用户体验(EUX)测试(FAT/UAT)	部署	异常响应
F3.2	基础设施 E2E 监控	工具选择	—	—	基础设施端到端测试(PAT)	部署	异常响应
F3.3	基础设施域监控	工具选择	—	—	基础设施域测试(PAT)	部署	异常响应
F4.1	信息流 E2E 监控	工具选择	标签信息	—	信息端到端测试(FAT/UAT)	部署	异常响应
F4.2	业务流程 E2E 监控	工具选择	标签/状态信息	—	业务端到端测试(FAT/UAT)	部署	异常响应
F4.3	终端用户 E2E 监控	工具选择	标签信息	—	真实用户监控(RUM)测试(FAT/UAT)	部署	异常响应

规划

DevOps 的规划阶段必须将调整监控工具所需的成本和时间考虑在内。免费工具也需要时间来配置。通常管理监控工具会被集中管理，一个新的敏捷项目可以充分利用现有的工具。然而，内建监控和应用程序接口监控需要在应用程序里编码。这些都需要在规划阶段就加以考虑。

编码

在编码阶段，必须知道如何通过监控工具捕获异常。在日志中记录一个事态（Event）并不足够，还必须使用某种能被监控解析的事态格式。最简单的方法就是定义标准、规则和指南（Standard, Rules & Guidelines, SRG）。标准必须满足；如果存在有效的理由，可以忽略规则；指南则推荐使用。

在编码过程中，以下最佳实践标准可以用来构建一个合适的监控工具：

- S1——每个事态有唯一的编号。
- S2——每个事态引用已经制定异常的软件配置项。
- S3——每个事态有一个已分配的级别代码。
- S4——每个事态定义了恢复操作。
- S5——每个新的事态会被输入运维团队的产品待办事项列表。

在构建阶段，应该知道哪些监控功能是可用的。服务台和运维团队是参与其中的重要利益相关方。

以下方面在编码阶段非常重要：

- 内建监控的功能必须通过编程的方式。重要的是，维护这些功能可能需要对应用程序进行新的部署。
- 事态监控要求在应用程序中正确的位置为事态编码。最重要的位置就是对另一个模块、另一个应用程序或基础设施服务进行外部调用的环节。
- 应用程序接口监控要求知道应用程序如何与其他应用程序通信。例如，可以实现一个控制，用以决定两个应用程序使用的通信协议是否为相同版本。
- 在只读/读写交易中，最好使用端到端信息系统监控。然而，对于读写交易，必须考虑在信息管理中这些交易不能产生脏数据。此时可以使用伪账户。无论如何，必须在整个应用程序中做到隔离，而且必须避免这样的伪账户的滥用。
- 信息流端到端监控要求数据格式可以度量。这就意味着必须为信息项贴上标签。最好在编码阶段增加标签。
- 业务流程端到端监控。为了监控业务流程，应用程序必须标记信息，用以确定交易属于业务流程的哪个部分。
- 终端用户端到端监控（真实用户监控）要求标记以确定哪个网络包会含有一种特殊类型的交易。

构建

在构建阶段，由于需要满足最长 5 分钟一次的构建过程，所以仅能使用单元测试。因此，完整的监控功能只能在稍后的部署流水线阶段进行测试。这就意味着大部分监控功能直到交付阶段才能测试。所以在编码阶段，进行尽可能多的度量以避免缺陷的产生是非常重要的。

构建时必须检查异常编码的格式，它们应该符合基于源码检查的标准、规则和指南（SRG）。在构建阶段，必须确认开发的每个错误信息已经记录在事态白名单或黑名单列表中了。

在编码阶段，程序员必须定义单元测试，在构建阶段将会使用这些单元测试，用来发现异常处理中的错误。所有误报和漏报都必须测试。

测试

只有在构建阶段创建了对象代码，在基线中检查了源代码之后，才有可能使用系统集成测试，以确认监控功能是否能在这个阶段发现异常。

通常认为，在部署流水线中的自动化验收测试仅需要涉及主逻辑。对此一个重要的例外就是，需要测试所有黑名单中异常的验收测试。

发布

一次成功的应用程序和基础设施组件的部署也应该含有对监控功能调整的部署，否则，新的事态将无法被识别。部署一个监控工具的调整通常没那么令人兴奋，然而，监控工具调整的回归测试是一个挑战。在实际工作中，这方面花费的时间非常有限。造成的结果就是，监控功能通常不如原来预期的有效。

另外，如果自动化部署不成功，部署流水线中的脚本也应该含有异常。因此监控功能也必须监控部署流水线，关注的不仅是可用性，还有异常现象。

运维

维护系统的正常运行，通常意味着服务偏差的恢复。因此，运维团队在确定需要监控哪些方面的时候，是关键的利益相关方。通常可以自动地解决偏差。由于开发和运维团队的协作，让在此提供最大支持变为可能。

监控

监控服务不仅对生产系统有用，而且适用于开发、测试、验收、生产整个周期。一个不能忽视的重要事实就是，实际工作中使用了越来越多的监控工具。它们应该彼此很好地通信。这些工具间的协作会影响编码阶段的工作。这是因为监控工具的通信，通常是基于编码阶段事态开发的格式的。

8

交付物（#07）

8.1 引言

本文介绍了一些 DevOps 规划的最重要的交付物。

8.2 术语

漏斗

漏斗是对选择路径的一种比喻，如图 8-1 所示。想法如此之多，以至于不能全部实现。随着一个想法变得越来越成熟，商业论证变得越来越清晰，这个想法就进入了漏斗，最后被选中成为路线图中的计划。实现使命、愿景、商业目标和实施策略是漏斗的主要驱动因素，而且，DevOps 组织要实现更快地将产品投放市场或更高的 SLA 指标也是漏斗的驱动因素。

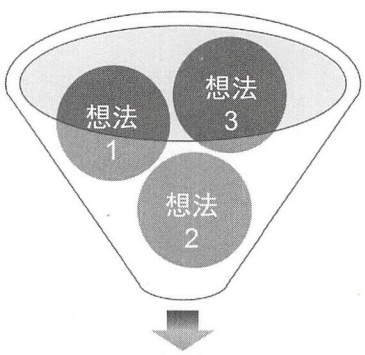


图 8-1　漏斗

路线图

路线图是描述产品或服务实现路径的一种方法（见图 8-2）。产品可以包含信息、应用程序或基础设施对象，用来为顾客提供服务。路线图显示了必须实现的最重要功能的时间表。必须考虑整个产品和服务组合，以避免产生瓶颈。

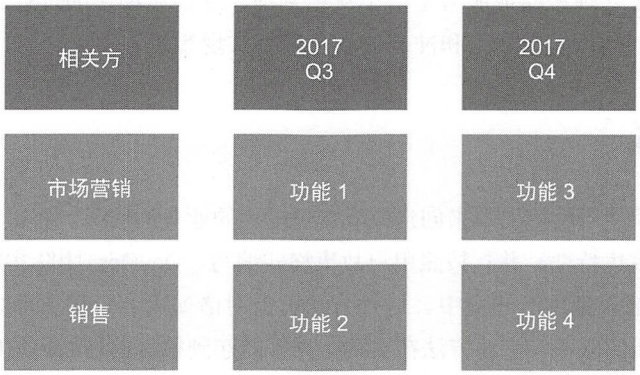


图 8-2 路线图

发布计划

发布计划是待办事项列表的基础。发布计划包含了必须实现的史诗。史诗代表产品组合中的项目。下一个版本将更详细地描述如何将史诗细化成特性（Feature）。

流

在 DevOps 中，经常使用精益原则，其中最吸引人的是“流”。DevOps 团队应该在零时间、零修复的稳定节奏中交付产品，这样，可以用最少的成本和理想的质量来实现最快的市场投放。

推和拉

在 DevOps 中，“拉动”机制的使用表明“流”是由需求而不是由供应控制的。因此，在供应范围内没有库存。

技术债

这个术语代表在实现阶段已经做出的选择，这些选择不是最优选项。这些产品和/或服务已经发布，但是需要进一步重新设计，以确保它们不会过时。在最坏的情况下，它是一种创可贴式的解决方案。

8.3 概念

功能分解

这个概念描述了实现产品或服务是一个从粗到细、逐步精细化的过程。按照这样的方式，通过漏斗、路线图计划、发布计划和冲刺计划，最终实现想法。

8.4 最佳实践

越来越多的组织使用敏捷原则来创建或改变对提供的业务的服务。在很多组织中，用待办事项列表作为起点来收集特性，并且按照用户故事格式编写。DevOps 团队实现这些特性，并且将创建和变更的对象部署到生产环境中。这种方法可能对诸如内容管理更新之类的小变更是有效的。然而，对于全新的服务，上述方法在实践中并不能起到很好的作用。这时需要基于体系结构的方法进行功能分解，通常这种方法不受待见，“完不成预定工作”（指计划不如变化快。——译者注）的说法常被用到。本文解释了基于漏斗管理、路线图和发布计划进行功能分解的好处。

8.4.1 完成预定工作

很多组织期望 DevOps 团队计划仅包含由产品负责人制定的产品待办事项列表。产品待办事项列表的顶部事项被细化，具有最高优先级。从顶部越往下，事项粒度越粗，优先级也越低。在这种情况下，经常用到“完不成预定工作”的说法。

在这些组织中，你也会看到，架构只会针对每次冲刺，并且架构是增量、迭代地制定的。只有一种解决方案的架构师负责指挥 DevOps 团队完成手头上待冲刺的变更，有时甚至没有专门的架构师。

对于很多简单的待实现的产品和服务，这种方法就足够了。对于涉及多个 DevOps 团队的复杂产品或服务，甚至一个已经组织起来的完整项目，再使用这种方法则不现实，必须使用功能分解。

8.4.2 漏斗管理

漏斗管理的一种反模式是：一个组织的开发团队在每个冲刺中都会收到新的任务或任务变更，因为这些任务必须快速完成。在这种情况下，不存在漏斗管理，甚至不存在“流”。事实上，

这样的组织不受源自使命、愿景、商业目标和策略的 DevOps 控制。即使存在一个目标方向，DevOps 团队与目标方向也毫无关联。

在这种情况下，认识到漏斗管理的好处是非常重要的，其主要优点在于：

- 漏斗管理定义了实现商业目标中非常重要的相关方，考虑了市场营销人员、业务经理、架构师、人力资源经理、产品负责人、安全官及服务人员。
- 漏斗管理要求在战略层面上反复考虑优先排序，并确保每个相关方感兴趣的内容发生变化时会被通知到。
- 相关方的合作能够保证对于变更的参与和承诺维持高水平。
- 架构师可以很好地指导相关方勾画出体现业务目标、战略和想法的目标体系结构。
- 能更好地刻画 DevOps 的想法可以包含在漏斗中。例如，用云端流水线部署代替本地部署（指本地自有服务器）。
- 漏斗管理提供了一种选择，根据方向来研发产品和服务，而不是基于每天的问题来开发产品和服务。
- 漏斗管理提供了通过确定成本和收益来定义商业论证的能力。这里的成本不仅包括 DevOps 团队的成本，还包括为缓解风险采取对策的成本。风险不仅与可行性和实用性有关（时间和金钱），还与 SLA 规范的实现有关。通过让相关方充分参与其中，就有可能让那些天马行空的想法变成有价值的思路。

对于某些组织而言，漏斗管理仅仅是个执行问题，有些组织还需要转型的策略。决定为什么要中断冲刺并避免流动是非常重要的。可以从区分两种类型的 DevOps 团队开始：一种 DevOps 团队仅受漏斗驱动，按照路线图和发布计划工作；另一种 DevOps 团队仅接受业务事件（Business-like Incidents）、救火事件（Emergency Cases）、临时工作等任务。这样做，至少可以让 DevOps 团队有机会证明 DevOps 的优点。

8.4.3 路线图

每个产品或服务都制定了路线图。制定路线图最简单的方法是用两个轴绘制一个图。纵轴是相关方，横轴是一年中的 4 个季度。接下来，每个单元都可以标识出相关方的哪些功能在哪个季度实现。

每个季度按块对功能和质量进行分类，这听起来很简单，但执行起来却是跨度很大的一步。从架构角度确认顺序是符合逻辑的，这一点非常重要。

总有些问题需要优先考虑，因为这些问题需要预处理，如安全认证、授权的形式配置、加密等，然而，通常这些都有标准特性可以使用。

产品和服务之间的接口并不相同，每个产品和服务都有自己的路线图，这些路线图的合集必须纳入一个协调的整体路线图中。如果一个产品或服务出现在多个链中，那么规划路线图将变得很困难。因此，没有体系架构的路线测绘和漏斗管理是不可行的。

8.4.4 发布计划

路线图建立后，发布计划很容易实现。未来功能的元素仍然模糊不清，可以在第一次发布之前详细描述实际的好处。虽然 DevOps 采用稳定流动的连续拉动机制，按照发布计划形式进行功能分解仍然非常重要。

发布计划的一种反模式是增量、迭代地生成应用程序，这将耗费更多时间且质量不断恶化，因为新功能不适应现有功能。通常没有更多的时间进行重构来重新设计旧产品和服务，发布计划只能按照路线图留出包括产品和服务重构的时间。

发布计划的另一种反模式是引入技术债。漏斗和路线图是一个好的 DevOps 流程和产品/服务架构的必备元素，从而避免了技术债。通过将架构贯穿于从想法到部署的整个生命周期，技术债的对策将被考虑在内。

9

瀑布式开发仍会存在（#08）

9.1 引言

本文介绍了采用瀑布式开发或迭代增量开发方法的一些观点及讨论。

9.2 术语

增量开发

敏捷的工作方法并非一次性构建一个完整的产品或服务，而是不断构建新增的功能。每个新版本都会增加一些新的价值，因此，其产品/服务的架构在设计时就需要考虑并支持其不断地灵活演进。

迭代开发

敏捷的工作方法是周期性地构建一个产品和/或服务。这些周期可能如敏捷开发中的 Scrum 一样是固定的，也可能类似于看板，每个交付所对应的周期不同。

瀑布式开发

瀑布式开发方法可以理解为一个完整的从需求到交付的产品交付过程流。它的步骤与 DevOps 流程（见文章#02）中的步骤相同，只是在瀑布式开发方法中，这些步骤只被执行一次。

9.3 概念

瀑布式项目

瀑布式项目采用瀑布式开发方法实现产品或服务。然而，为了开发和实现这个产品，瀑布式开发项目可能包括多个阶段。如果采用阶段式开发，产品功能描述中的分解功能结构在前期通常都是比较粗糙的，要在后续每个阶段逐步深入细化。

敏捷项目

敏捷项目始于一个商业论证、路线图和发布计划，通过持续交付增量的产品或服务得以落地。在周期迭代过程中，产品待交付列表中的功能点被逐步完成，并在路线图和发布计划中明确定义优先级和顺序，但也可以按需在必要时进行调整。通过周期迭代，每期产品都确认是可上线的。同时，这种方法也可以验证之前的商业论证在当前情况下是否依旧有效，或者另一个项目可以用更少的资金和时间为业务提供更多价值。

9.4 最佳实践

当准备采用敏捷方法进行项目管理时，一个最重要的讨论就是敏捷方法是否适用于所有项目。敏捷方法要求产品和服务应该采用增量和迭代的方式进行构建，而且每个增量都应该为业务带来价值。DevOps 则更进一步，声称“完成”就是“能在生产环境中运行”。然而，在大部分组织里，20%的项目仍然正在构建单体解决方案（monolithic solutions）。这意味着由每次迭代带来的增加价值只能在完成整个项目后一次性实现。尽管如此，在这些项目中采用 DevOps 方法依旧可行，只是在每个周期中增加价值的思想和初衷将失去意义。因此，DevOps 团队会感觉在这些项目中使用 DevOps 方法并不合适。本文给出了在新项目中是采用 DevOps 方法还是瀑布式开发方法的建议。

9.4.1 瀑布式开发会一直存在

从一份对 10 个机构的调查[Best 2015a]结果中发现，在所有已经采用敏捷项目管理的组织中，仍然有瀑布式项目存在，而敏捷与瀑布式项目的数量比通常是 80：20。其实，这两种方法都各有优势。关于敏捷和瀑布式项目的选择方法，以下所述是具体的标准。

9.4.2 敏捷项目管理适用之处

- 信息系统中功能和质量要求没有明确说明（定义不明确）。
- 需要快速响应投放市场的要求。
- 每个增量都须增加价值。
- 信息系统需求及功能灵活多变。
- 在组织内需要持续优化。
- DevOps 团队能够对信息系统做出快速和适当的调整。
- DevOps 团队能够较好地处理迟到的需求变更。
- 需要满足 DevOps 开发流程更经济的硬性要求。
- 开发人员非常资深且可以独立工作。
- DevOps 开发团队具有自我管理的能力，能适应多变的环境。
- 采用 DevOps 开发的信息系统本身为松耦合设计。

9.4.3 敏捷项目管理不适用之处

- 组织不希望或不能发生改变。
- 所需的知识和技能储备不足且相关人员也没有欲望进行能力拓展。
- DevOps 团队成员缺乏经验。
- 组织需求是预设且稳定的，对于项目实施周期和成本的需求也是如此。
- 信息系统的适用环境是稳定且很少变化的。
- 信息系统和其他系统是紧耦合的。
- 由于 Scrum 开发流程与强规则（de Caluwé[1] Blue）绑定，组织不希望自控制和自开发（de Caluwé Green）。
- 从分析到实现之间的要求不会或预计可能不会发生变化。
- 组织中思考和工作方式是高度等级化的，项目经理担当了产品经理的角色，他可以像管理一个瀑布式开发团队那样管理一个敏捷开发团队。

9.4.4 实质

DevOps 的核心理念是“完成”，即用户在每个迭代周期结束后得到一个可运行的产品，其

1 de Caluwé 是一个把变革用 5 种不同颜色表示的模型。——译者注

为业务流程带来增值。然而，许多组织在产品经历了多个迭代周期后才开始采用 DevOps 方法去挽救最终结果。这种方法本身并没有错，但却意味着产品推迟上线及反馈环滞后。

瀑布式开发项目往往存在新增价值实现较晚的缺点，而这正是与敏捷项目的不同之处，因此其投资回报也就相对较低。

10

从漏斗到 Scrum 板（#09）

10.1 引言

本文描述了 DevOps 语境中的一些关于需求的最佳实践。在本文中使用的术语有可能和互联网上或其他方法论中的定义不同。由于这些术语没有很明确的定义，它们又对本篇文章的意义重大，因此将这些术语的定义放在文章的前面。事实证明，这些术语很容易嵌入工具中且易于学习。

10.2 术语

主题

主题描述了通过敏捷项目交付的创新。一个主题可以类比为一个项目，并且包含一个以上的史诗。主题的生命周期开始于漏斗中的一个想法或创意（参阅文章#07）。

史诗

史诗是对包含多个 DevOps 周期的一组功能的描述，常用于定义路线图。史诗是主题的构建块，可以再被分解成特性。一个史诗可以类比为一个项目的阶段计划。

特性

特性是对一个功能的描述，其所需开发时间通常少于半个生命周期。特性用业务术语来表述，是史诗的构建块，与故事有所区别。

故事

故事是对一个要实现的对象或对象的配置的技术性描述。故事可以类比为变更请求（RFC）。

用户故事

用户故事是一种描述主题、史诗或特性的协议（protocol）。用户故事的格式如下：

我作为<角色>

想要<需要>

以便<增加价值>

需求

需求是对加诸于信息、应用、基础架构或服务的一种需要。需求可以是功能性需求，也可以是质量需求。质量需求也称作非功能性需求（NFR）。

测试类型

下列测试类型经常被使用：

- UT = 单元测试
- ST = 系统测试
- SIT = 系统集成测试
- FAT = 功能验收测试
- PAT = 产品验收测试
- PST = 性能压力测试

10.3 概念

可追溯性

DevOps 是一种可以加快产品进入市场的方法。对于关键的业务服务，重要的是要快速找到故障及修复它。然而，找到前面的版本并修复它，并不总是能做得到的，因此可追溯性的概念与 DevOps 的关系非常大。

可追溯性的含义是当在生产环境中发生了一个事件，可以追溯到原有的功能或质量的变更。

这就要求：

- 在生产环境中的对象具有唯一的标识（ID）。
- 生成的事件包含元数据，元数据提供了事件发生源的对象 ID，可以快速洞察到。（参阅文章#06）。
- 每个唯一标记的知识都记录了最新的修改（构建 ID）。
- 在每个构建中，底层的源文件是可识别的。
- 在所有源数据文件中，很清楚哪些静态需求是适用的，以及这些源文件对应的测试结果是什么。
- 需求总是与一个特性联系在一起的，并且适用于史诗和主题。

这种形式的可追溯性指的是向后追溯。如果需求发生变化，可以使用向前追溯来确定哪些对象需要修改。

可复用性

对于一个组织来说，功能的可复用性非常重要，因为它节省了大量的时间。此外，功能的可复用性对质量控制也很重要，因为现有的对象通常比新对象更稳定。通过提供元数据需求，它们更容易被检测和使用。

10.4 最佳实践

开始 DevOps 的旅程首先从学习其基本原理开始。在一段时间之后，人们通过不断学习让 DevOps 流程更加成熟。这是一个很自然的工作方式，是每个 DevOps 团队自下而上自我发现的方法，是一个团队自我学习的典型例子，即使一些组织机构选择不再设直线经理。另一种方法是从架构的角度来看待 DevOps 组织的形成。我们希望在哪里结束？定义一个粗略的目标架构的好处是我们可以定义过渡计划和里程碑。这个目标架构并非不可更改，只是一个粗略的描述，因此每个 DevOps 的团队在它们各自的发展进程中依旧有它们自己的学习曲线。重要的风险可以得到很好的管理，投资可以有更好的前瞻性。在此情况下，主要关注围绕着 DevOps 团队的“可追溯性”的信息架构，目的是为了阻止信息债。

10.4.1 生命周期管理

需求的生命周期开始于在漏斗中建立一个主题，在那个时候功能和质量需求还没有成形。主题通常不以用户故事的形式来描述。在产品路线图的起草过程中，功能和质量需求变得更具体，然后把主题分解成史诗。功能和质量需求以用户故事的特点和格式来表达。然而，产品特性只是以业务术语来描述可交付成果。最后，转化成故事这一过程提供了陈述技术需求的机会。

10.4.2 功能和技术设计

筛选过的符合验收标准的功能可用于创建功能验收测试（FAT）和用户验收测试（UAT）的计划。由于这些功能是以业务术语定义的，它们不适合形成产品验收测试计划，因此，特性需要分解成故事。

有一点很重要，就是特性不能取代功能设计，故事也不能取代技术设计。有时候一个特性指一个功能设计，而故事则参考一个技术设计。但是，这不是推荐的方法，因为工具中描述的特性/故事与描述实际规格的功能/技术设计文档之间存在脱节现象。

更好的做法是为特性配备一组功能和非功能性需求，使特性成为具有自己标识的独立对象。在大多数工具中，可以将需求链接到特性。定义技术设计要求的故事也是如此。

10.4.3 测试用例

必须将功能要求和技术要求与测试用例联系起来，以确保相关方的意图（特性）和技术需求（故事）的实现。

10.4.4 可追溯性

DevOps 提供从需求到部署的快速反馈周期。服务中断时必须尽快与新的或变更的需求（向后追溯）相联系，但是新的或变更的需求对已经部署的功能的影响也是非常重要的（向前追溯）。

为了实现这两种形式的追溯，从主题到生产环境中的配置项（CI）都必须有良好的对象注册。在图 10-1 中，绘制了一个对象模型和它们之间的关系。在选择 DevOps 工具时（参阅文章＃05），要注意哪些工具包含哪些对象，以及如何在需要的地方将它们链接起来。

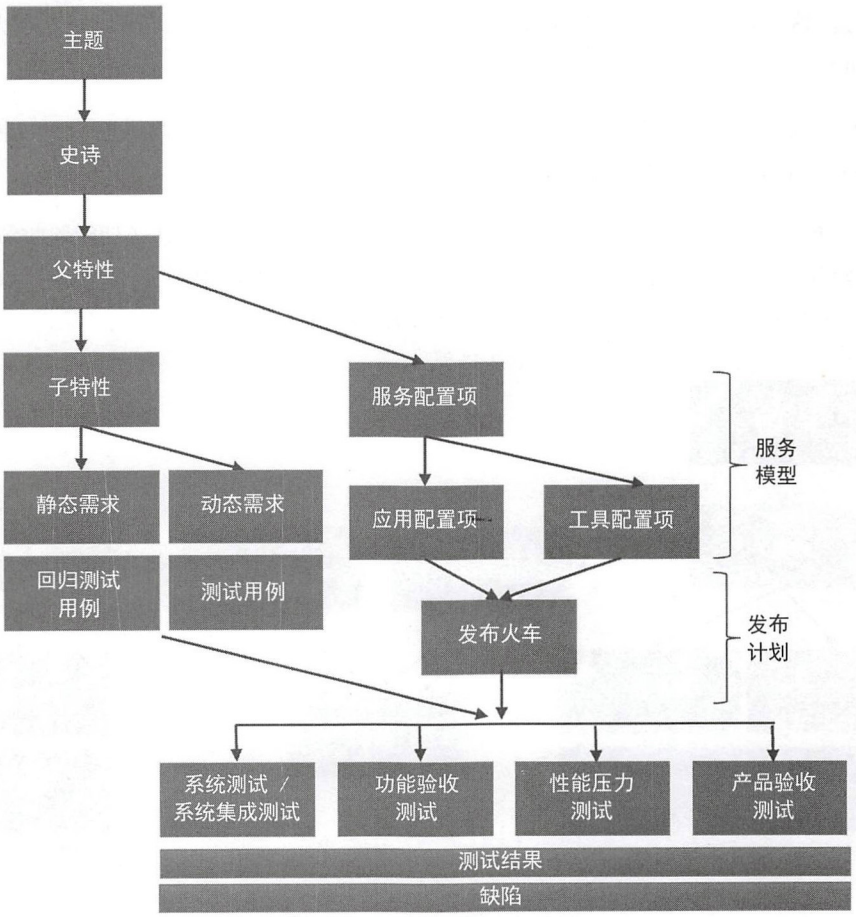


图 10-1　从漏斗到故事看板

可追溯性从主题开始，一个主题包括一个或多个史诗。上述情况也适用于史诗和特性的关系。原因在于特性可以按功能被划分为几个片段，所以也可以构建 1∶*n* 的关系。由于一个特性会用业务术语来描述，因此软件的配置项（S-CI）和功能之间的关系最为明显。

特性和需求也是 1∶*n* 的关系。需求也许是临时的，仅适用于功能实现的周期（动态），或者是持续的并在功能的整个生命周期中都是必需的（静态）。静态需求对应回归测试用例，动态需求对应普通测试用例。

服务配置项由一个或多个应用配置项和工具配置项组成，合在一起成为将服务划分为技术组件的服务模型。为了保持其可读性，基础架构的配置项被忽略掉了。对象的部署最好以固定

的速度进行，这就是为什么将对象部署比喻成火车，把一个配置项放置在一列发布火车上以推出（roll out）它。

服务配置项是否部署在发布火车中，取决于服务配置项的质量。测试取决于各种不同的测试类型（ST，SIT，FAT，PAT，PST）。

ST（系统测试）和 SIT（系统集成测试）发生在测试环境中，FAT（功能验收测试）、PST（性能压力测试）和 PAT（产品验收测试）发生在验收环境中，测试用例与需求及对象相关联（应用配置项或工具配置项），最终，测试结果同测试用例和测试缺陷联系起来（见图 10-2）。

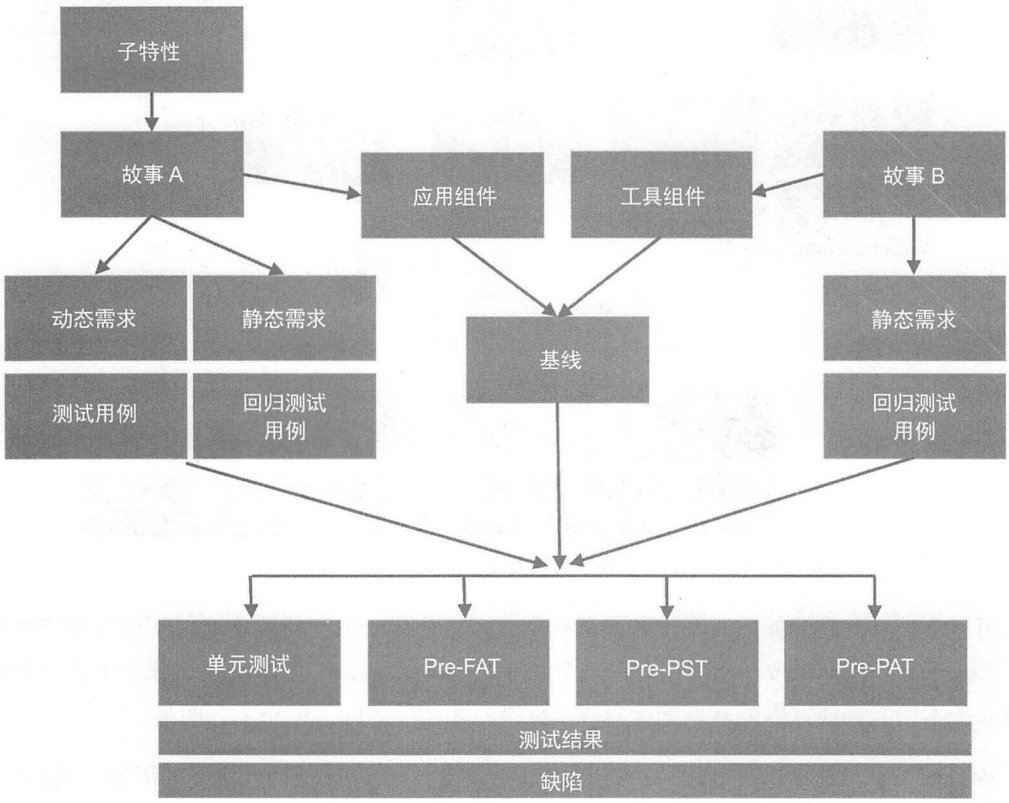


图 10-2　从漏斗到故事看板（详细视图）

图 10-2 是比图 10-1 更详细的视图。在这里故事都与特性相关。另外，测试类型颗粒度低。这里的 FAT/PAT 和 PST 都加了前缀“预”（Pre），这标志着这些测试并不是实际的 FAT/PAT 和 PST，而是早期在验收环境中的测试。

10.4.5 信息债

很少有组织认识到的一个非常重要的风险是信息债。信息债是在敏捷项目中的一种现象，是由于对象的元数据管理不足而导致的，如图 10-1 和图 10-2 所示。

这一问题造成将被部署到流水线的工具集几个月内都无法正常工作，其原因是没有记录元数据。这将导致手工测试或测试不完善。因此，可以说大部分 DevOps 团队目前都在生产遗留物（指没有办法做到全面的自动化测试。——译者注）。

11

服务级别协议和非功能性需求（#10）

11.1 引言

本文描述了在 DevOps 方法中如何定义服务级别协议（SLA）。SLA 包含服务的非功能性需求（NFR）。

11.2 术语

SLA 是基础设施技术（IT）服务提供者和客户之间的协议。SLA 的内容是关于产品、流程和服务的质量标准。一个应用程序的 SLA 规范的示例是：可用性 99.9%，最多每年 3 个事故和最长 3 小时服务中断。在本文中，术语中产品包括服务。

11.3 概念

SLA 的一个重要概念是：服务标准必须来源于业务流程目标，如图 11-1 所示。

业务流程（3）使用产品和服务（5）。SLA 标准（6）的关键成功因素（CSF）（4）可由关键性能指标（KPI）（4）进行度量。重要的是要认识到，CSF 是业务目标（1）不能达成的风险对策。可以通过技术产品和服务（16）的实现和服务管理流程（14）来减轻部分业务风险。事实上，SLA 是关于供应商应对措施有效性的协议，用来降低客户的业务风险。

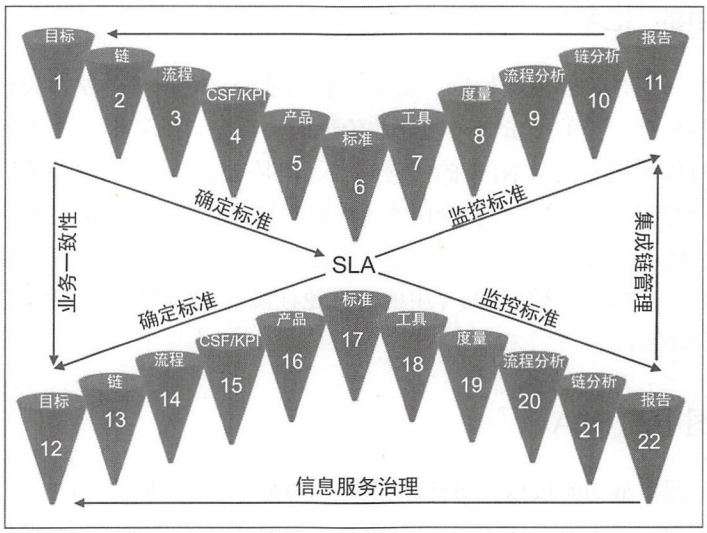


图 11-1　业务一致性

11.4　最佳实践

本文将解释 DevOps 环境中 SLA 的作用。首先，初步拟出 SLA 的商业论证；其次，定义 SLA 在 DevOps 过程中每个阶段的影响。

11.4.1　是否还需要 SLA

如果市场采用一个新的方法，许多现有的控制措施将被废除。遗憾的是，引入 DevOps 也是如此。很多组织将 SLA 当作一种昂贵的文档，认为对服务交付的质量没有任何影响，这是对 SLA 实际目的的严重偏见。

SLA 可以作为测试的基础，以确定服务组织可以在多大程度上交付什么样的质量。如果 SLA 里定义的质量标准从未被满足过，这说明服务级别经理（Service Level Manager）的能力和服务组织的成熟度还不够，而不应置疑 SLA 本身是否有用和有必要；如果 SLA 不包含任何质量标准，这说明客户需求不清晰、业务流程治理的成熟度不够，或者供应商对如何满足客户需求知之甚少。

因此，SLA 只不过是一个从业务角度合并非功能性需求（NFR）的文档，没有理由说 SLA 没有存在的必要，只能说信息载体（一份文档）现在已不够了。

11.4.2 漏斗中的 SLA

服务标准，如可用性、容量、性能、安全性和连续性，对服务的整体设计都有影响。这些标准及对满足这些标准所需的度量必须是商业论证的一部分，在漏斗中加以确定（见文章#07）。人们常说在那个时候还没有关于 NFR 的任何想法，但是，这个说法站不住脚。如图 11-1 所示，一个 NFR 就是从业务到风险控制的一个问题。采取行动需要额外功能，这极大减少了漏斗中商业论证的想法。然而，高级别的 NFR 将被包含在这个想法里（这是指在商业论证中应包含高级别的 SLA，即一些 NFR，而这会使漏斗中的商业论证减少，因为实现这些 NFR 需要对应更多的软件功能。——译者注）。

11.4.3 路径图中的 SLA

在 ITIL®中，服务战略阶段以业务活动模式（PBA）和用户配置（UP）的形式提供了一个极好的框架。架构师应该和相关方及产品负责人一起识别服务的使用者有哪些，以及他们希望从 PBA 中获得什么。这样，负责操作的员工（operational employee）通常会高频率地使用服务，负责策略的员工（tactical employee）则使用频率较低。特别是，必须更密切地考虑基础设施上的服务占用，以确定对基础设施的影响。随着云服务的到来，这些影响很快就会减弱。尽管如此，我们还是应该密切关注这一点。其他方面，如安全性，是一个持续的主题。

11.4.4 发布计划中的 SLA

非常明显的是，正确发布所需要的基础设施变更常常被遗忘。对于组织来说，确定基础设施是 DevOps 团队职责的一部分（基础设施即代码）。由于服务的性质，集中管理的基础设施团队经常使用瀑布式方法。基础设施团队必须及时参与，而不是在上线前三天才参与。

11.4.5 编码阶段的 SLA

程序员常说 SLA 是为运维（OPS）团队设计的，对他们的工作没有任何积极或消极的影响。当然，这是错误的。架构师必须密切关注 DevOps 的每个周期，或者要对服务性能产生积极的影响。至少必须使用一个简单的检查清单，例如：

（1）可扩展性（这个软件可扩展吗？）

（2）性能（查询计划是否显示表扫描？）

（3）安全性（是否需要加密，安全策略是否适用？）

（4）可用性（异常处理是否得到很好的实现？）

（5）数据质量（使用的是正确的处理机制吗？）

11.4.6 构建阶段的 SLA

在构建阶段，单元测试用例可以检查与物理资源的优化使用有关的标准规则和指导原则（SRG）是否被遵守了。例如，内存泄露可以被检查，同样，代码冗余也应被检查。性能测试只发生在构建之后，我们不应该忘记，构建服务器也应该提供良好的性能以实现准时上线。一个黄金标准：一次构建不超过 5 分钟。但仍然有一些组织需要花整晚时间进行构建，这会导致很多不良后果。

11.4.7 测试阶段的 SLA

测试阶段对于非功能性需求（NFR）的测试自然重要。例如，确定黑名单事态（Blacklist Event）上的所有事态是否像在生产环境中一样被监控器感知。而且，回归测试特别重要。智能监控设备正越来越多地被用于测试阶段（参见文章#06）。

11.4.8 发布阶段的 SLA

部署流水线应该是快速且可重复的。当然，生产环境在被严重干扰的情况下可恢复性这一需要，要求部署流水线必须达到高标准。在这里架构也扮演了重要的角色，以确保不使用错误的构建机制，并保证工具间的连接。

11.4.9 运维阶段的 SLA

服务级别在运维阶段得到最终检验，但希望 SLA 标准已经在漏斗中进行评估，并在测试阶段进行了测试。运维阶段的反馈会让我们对系统做定期调整，同样，这些调整也会经历从想法到部署整个流程。

11.4.10 监控阶段的 SLA

所有 SLA 标准都应该通过监控工具来观察，这对监控设施提出了很高的要求。不能被监控的也不应被同意（指不应同意的 SLA 已经被满足了。——译者注）。

因此，SLA 对 DevOps 过程中的所有阶段都有影响。事实上，让 DevOps 团队编写一个维护水平协议（OLA）是非常明智的，他们应在其中写明作为一个团队如何在流程、产品、服务中定义对策以保障 SLA 标准，如图 11-1 所示。

12

功能和技术设计（#11）

12.1 引言

本文描述了功能和技术设计在 DevOps 环境下扮演的角色。

12.2 术语

特性

一个特性就是耗费不超过半个 DevOps 周期时间（开发）的一个功能。特性是用业务术语来描述的，是史诗的构件块，和故事是不同的。

故事

一个故事是一个待实现对象或对象配置的技术描述。一个故事就相当于一个变更请求（RFC）。

功能设计

功能设计描述了需要构建什么。功能设计通常涵盖一个应用程序的以下几个方面：

- 功能模型
  - 业务流程
  - 呈现服务
  - 流控制

- 连接
- 数据访问功能

- 组件模型
- 信息模型
- 工作流控制
- 流程模型
- 系统使用
- 非功能性需求

技术设计

技术设计描述了如何构建应用。技术设计通常包含以下几个方面：

- 技术模型
  - 硬件
  - 操作系统
  - 连接
  - 系统管理
  - 数据库管理
- 组件模型
- 实现模型

12.3 最佳实践

产品待办事项到底能不能立即开始编码加以实现？这是DevOps新手团队经常讨论的话题。为什么还需要做功能或技术设计呢？这些设计既耗钱，又从未能保持及时更新。即便这些设计是必需的，这些设计又如何适用于增量、迭代开发产品或服务的思路呢？而且，如果客户经常删除特性或增加特性，总是改产品待办事项列表，该怎么办呢？功能和技术设计的这种理念看样子已经落伍了。本文探讨了功能和技术设计的虚虚实实，以及它们与特性和故事的关系。

12.3.1 DevOps 流程

DevOps 流程并没有安排一个设计阶段。编码阶段从规划阶段就已经开始了。这也是众多DevOps 团队所做的。从特性开始，创建一个数据库，用 Java 写个功能，再执行一次构建。跳

过那么多步骤，是因为它们都已经包含在功能和技术设计模板里了。

问题是这样做有没有错，或者说是否明智。答案就在于功能和技术设计对 DevOps 团队来说到底有什么价值。

12.3.2 功能设计

以下这份清单展示了默认的附加价值。放到 DevOps 环境下，逐个评估它们的有效性，就能够查明功能设计还有什么剩余价值。

功能设计作为交付物的商业论证如下：

（1）我们需要思考一下应用程序和业务流程之间的关系，这样就能对变更产生的影响有所洞察。

（2）工作是一个由粗到精的过程，而不是循环往复地简单堆叠独立的功能。

（3）应用程序的应用信息转换（输入处理—输出）能够匹配应用程序所支持的信息服务。

（4）已有用户配置画像和业务活动模式信息用于确定非功能性需求（NFR）。

（5）数据模型已经做了映射，防止出现同音异义词和同义词的情况，以此预防与其他应用间的接口问题，以及因不正确地汇总不同定义的数据而导致企业报告错误的问题。

（6）确定紧急情况下所需要的最小数据集和功能。

（7）弥合一个时间段。如果一段时间后需要修改应用程序，程序有功能文档可用就很重要了。而且，功能设计可以成为不同 DevOps 团队之间的一种沟通手段。

建议 1 对 DevOps 团队来说，这部分功能设计仍然很相关，必须在漏斗和路线图规划中定义出来。DevOps 团队也能受益，因为从中可以了解邻近应用程序（adjacent applications），在上线前进行一个链式测试（chain test）。

建议 2 对于 DevOps 团队来说，粗粒度分解已经在路线图里的主题（theme）体现了（参见文章#07）。

建议 3 这部分功能设计无法提前定义。它依赖在 DevOps 周期循环过程中所做的选择。功能设计可以靠特性的实现来补充，这是对完成的定义（Definition of Done，DOD）的检查。

建议 4 提供用户配置（UP）/业务活动模式（PBA）是商业论证的一部分，它们是以 DevOps 方法开展敏捷项目的先决条件。

建议 5 这部分功能设计无法提前定义，只有信息区域和信息类型可以提前确定。基于此，和其他应用之间的接口也可以被预定义（应用程序全景图）。不过，数据模型会随着 DevOps 循环中所做的决定而变化，因此数据模型必须是循环演进的，这是对完成的定义的检查。注意，数据库管理多年来一直声称，预先定义数据模型是开发一个运转正常的应用程序的必备条件，而且对于增量式开发的应用程序来说也是如此。因此，扩展数据模型也就成为头号风险。对策是重新设计数据模型，使其对应的每个需要的应用程序版本都有带回滚场景的数据迁移。这种改造的成本很高，回滚场景也很容易出错。数据模型预先映射得越好，性能下降的风险也就越小。

建议 6 在 DevOps 的世界中，应急管理（Contingency Management）是关键，注意力要聚焦在最少需求信息（Minimum Required Information，MRI）集的应急处理上。然而，我们必须记住，大多数组织的数据处理都非常复杂，以至于很容易在遇到应急事件时要么什么都做，要么什么都不做。若有必要，可以把应用程序分类为关键业务或非关键业务。基于此，你就可以确定是为某个应用程序实施应急度量还是都不处理。

建议 7 产品待办事项列表无法填补这个角色的空缺。已实现特性应该已经被保存，但是，与那些未完成特性摆一块，它们就成了彼此没有相关性的对象，结果，事物的一致性丢失了。

基于上述分析，功能设计有些部分很适合对应用程序进行自上而下分析（漏斗、路线图、发布计划）。功能设计的其他组件应该随着应用程序的逐渐实现而演进。最好每实现一个特性就做功能设计。所以，把特性拆成故事后，就必须考虑更新功能设计。只有这样做，才能让文档保持最新状态，并留意潜在的风险，以便在特性实现的过程中加以控制。DevOps 的“完成”意味着功能设计已更新。

然而，问题来了，是否每个应用程序都应该有功能设计呢？答案是否定的。但只要有了漏斗、路线图和发布计划，就必须有功能设计。因为特性是用来描述“是什么”的，而它或许能确保特性是功能设计的一部分。事实上，特性就应该与元数据一起提供，以便用于功能设计。如果把功能设计以文本形式存到代码库里，再加个版本号，就可以基于库和产品待办事项列表来生成功能设计了。因此，功能设计就与目标代码一样，作为制品存在制品库里。

但也有些应用程序并不需要功能设计。有没有它们，特性都一样可用，它们自身就描述了功能设计。为了保持一致性，给这些应用程序也生成功能设计会很有效。

12.3.3 技术设计

技术设计作为交付物的业务场景是：

（1）需要探讨应用程序和框架之间的关系，在本地服务或云服务上都需要些什么。

（2）需要一张基于用户配置和业务活动模式呈现基础设施利用率的图。派生出的 NFR 必须确保满足 SLA 标准。

（3）需要一个可用的实现模型，用以描述所需的维护技能和任务，以及要实现的各个方面。

（4）弥合时间。假如过了一段时间，需要整改某个应用程序或某个基础层功能，有技术文档可供查证以判断其影响就很重要了。而且，技术设计可以成为不同的 DevOps 团队之间的沟通手段。

建议 1 应用 Docker 和云服务等方法越来越常见。Amazon 已经推出了消息队列和 NoSQL 数据库的应用程序接口（API），大大减少了对详细阐述的需求。然而，如果设计师基于漏斗开始进行功能设计，并在路线图和发布计划中进行扩展，就应该做技术产品（基础设施、系统软件、工具等）的设计。

建议 2 有了基础设施即服务（IaaS）和平台即服务（PaaS）后，NFR 已不如以前那么重要了。安全是另一回事，但仍然有必要仔细观察信息被存储在基础设施中的位置，以及这样做是否符合法律法规。

建议 3 如果要对基础设施强加安全评估和权限等其他需求，就必须调整发布流水线。必须及时了解这些信息，以避免干扰 DevOps 团队流畅地工作。

建议 4 与功能设计一样，分享知识很有必要，这种知识分享还必须弥合时间和距离。经过前述分析我们发现，就像功能设计，技术设计也有一部分很适合对应用程序和基础设施的使用进行自上而下的分析。技术设计会随着每个特性的逐渐实现而演进，因此，特性只要被拆成了故事，就必须关注技术设计的调整。只有这样做才能让文档保持最新状态，并留意潜在的风险，以便在特性实现的过程中加以控制。DevOps 的“完成”意味着技术设计已更新。

故事反映了特性的技术实现，所以，技术设计必须基于库里与故事相关的文本对象生成。技术设计也必须作为制品存在制品库里。

12.3.4 结论

功能设计和技术设计是应对不可忽视的风险的策略，然而，这些设计应在基于漏斗的路线图和发布计划期间确定下来。功能设计的细节基于特性而获取。特性必须以元数据属性来描述，这样它们才能被生成和被纳入制品库。功能设计的扩展应该基于特性而非循环来开展。此规则同样适用于技术设计，只不过技术设计基于故事而非特性。

13

分解特性（#12）

13.1 引言

本文讲述了分解特性的两种方式——垂直方式和水平方式，以及这两种方式的优缺点。

13.2 术语

特性

一个特性就是耗费不超过半个 DevOps 周期时间（开发）的一个功能。特性是用业务术语来描述的。特性是史诗的构件块，和故事是不同的。

故事

一个故事是一个待实现对象或对象配置的技术描述。一个故事就相当于一个变更请求（RFC）。

13.3 概念

分解特性

有两种跨 DevOps 团队分解特性的方式——垂直分解和水平分解。垂直分解指特性由单个 DevOps 团队实现；水平分解指特性由多个 DevOps 团队协同实现。

特性最终被分解成待实现的故事。垂直分解意味着某个特性由单个 DevOps 团队实现，所以其分解出来的故事也由该 DevOps 团队处理（见图 13-1）。

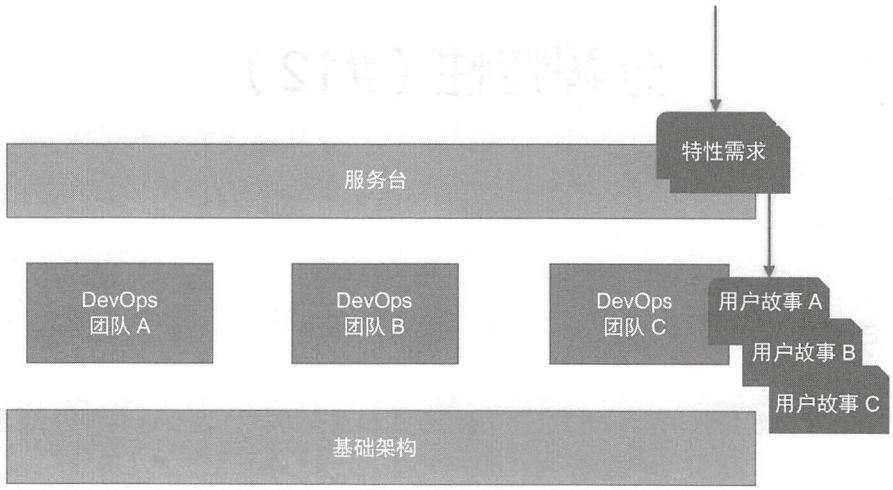


图 13-1　垂直分解特性

水平分解特性意味着一个特性由多个 DevOps 团队共同实现，所以其分解出来的故事要分给各个 DevOps 团队处理（见图 13-2）。

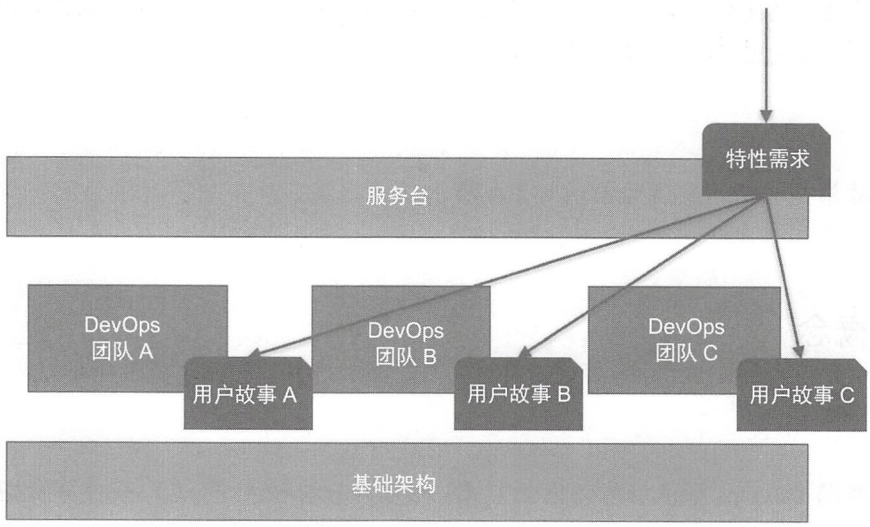


图 13-2　水平分解特性

13.4 最佳实践

大型复杂的应用程序通常需要更多的 DevOps 团队一起工作。 理想情况下，这些 DevOps 团队可以独立工作，这样可以通过减少浪费（相互等待的时间）来缩短上线时间。 但是，有一些情况还是允许多个 DevOps 团队在一个特性上共同工作的。本文介绍了两种特性分解形式，并最后给出了建议。

13.4.1 垂直分解

垂直分解的优点在于，在特性的生命周期中，从编码开始到生产环境的监控不存在依赖关系。因此，DevOps 团队必须具备实现该特性所需的知识和技能。

垂直分解的优点具体包括：

- 加快上线速度。因为团队实现特性的整个周期计划不依赖其他团队。
- 完成高质量服务。因为团队实现了完整的工作流，不需要其他 DevOps 团队协助同步开展测试工作。
- 组织更容易扩展。因为 DevOps 团队可以轻松扩充。
- 产品的待办事项易于管理。因为只有一个产品负责人和一个 DevOps 团队负责此事。

垂直分解的缺点是可能有多个 DevOps 团队在不知道彼此的情况下调整应用程序，这样可能在变更中引发事件、产生损失。例如，5 个独立运作的 DevOps 团队开发内容管理系统（CMS）。如果这 5 个 DevOps 团队只关心自己要实现的特性，不考虑其他 4 个 DevOps 团队所负责的特性的影响，可能引发其他 4 个 DevOps 团队产生事件，甚至可能导致其他 DevOps 团队发生变更。

13.4.2 水平分解

水平分解的优点在于 DevOps 团队可以专注于其拥有丰富知识的相关服务，与其他 DevOps 团队一起协作，共同具备实现整个服务的能力。

水平分解的缺点包括：

- 由于多团队之间需要协调、交互，降低了市场响应能力。
- 即使各团队专业性强，实现故事的质量很高，也只有各团队将各自的故事和测试阶段匹配得很好的时候，才能保证服务质量。
- 团队拥有特殊的知识，使得团队不容易扩展。

- 团队的管理变得非常困难，需要征求多个产品负责人的意见。
- 相对于垂直分解，由于发布包含的组件由多个团队交付，上线决策非常困难。评估一个不能工作的组件所造成的影响也非常困难，因为这需要对所有产品都非常了解。

13.4.3 建议

垂直分解是优先推荐的，然而，某些情况下垂直分解几乎不可行。例如，一个组织从一个供应商处买了若干产品且需要大量配置工作，每个产品的学习周期都很长（超过一年），垂直分解需要团队掌握所有产品的知识和技能，这显然不合适。在这种情况下，水平分解不可避免。此时这些 DevOps 团队需要有了解各自负责的产品的各个方面的专家。

14

定义特性和故事（#13）

14.1 引言

本文主要描述如何定义特性和故事。

14.2 术语

特性

特性是对功能的一种描述，每种特性所需要花的时间不应超过半个 DevOps 周期时间。特性是用业务语言描述的，是史诗的构建块（building block），和故事是不同的。

故事

故事是对要实现的对象或对象的配置的技术性描述。一个故事类似于一项变更请求（RFC）。

14.3 概念

数据建模

数据模型指的是对一组需要被管理的信息的描述，如特性和故事。对于数据模型来说，重要的是数据对象中可用的数据元素，它同时也描述了对象之间的关系。

信息债

这个概念表达的是在部署中有无法使用的最优化的流水线工具，因为在过去，对工具中使用的对象提供的元数据太少。例如，如果不清楚哪个测试用例属于尚未更改的特性和需求集，那么测试机器人就不能为回归测试选择测试用例。

14.4 最佳实践

为了能正确地管理 DevOps 的对象（如特性和故事），非常重要的一点是从用户那里了解他们对这些对象的信息方面的需求是什么。这些数据必须被建模，而且需要让它们彼此之间形成关系。本文描述了这两者（特性和故事）的各自属性以及它们之间的关系。

14.4.1 为什么需要数据模型

DevOps 方法中一个重要的方面是部署流水线。为了让流水线实现最大的吞吐量，需要许多工具协同工作。工具的协同除了把工具连接好，一定程度上还依赖工具能否提供正确的信息。因此，将相关方需要的信息对应到 DevOps 流程上是非常重要的。与此相关的另一个方面是信息债。

14.4.2 特性的属性

下面这个清单显示了在针对特性的管理中应该从哪些方面入手。

- 特性 ID
- 史诗 ID
- 申请人 ID
- 申请日期
- 优先级
- 故事点数
- 期望的发布火车
- 确定的发布火车
- 关联的持续集成应用程序
- 关联的持续集成服务
- 单行用户故事

- 状态
- 审计跟踪

特性 ID

特性 ID 是非常重要的，因为有许多对象会与这个特性相关联。例如，一个需求、一个用户故事及一个功能验收测试用例都包含特性 ID。

史诗 ID

如果一个特性源自某个用户的问题，则可以不需要父史诗；但在实现产品和服务的过程中，会需要一个父主题及史诗。这个字段就描述了这样的关系。

申请人 ID

一个特性的提出者必须被告知特性的状态。这就意味着一个特性会关联一个 ID，如一个活动目录（active directory）ID。用来描述这些特性的工具必须有一个轻量级目录访问协议（LDAP）来做链接。

申请日期

定义好提出需求申请的日期非常重要，因为最长的响应时间及实现时间应该已经在 SLA 里定义过了。

优先级

关于某个特性的优先级取决于该特性的商业价值及实现它所需要的工作量。商业价值由产品负责人决定，所需工作量由 DevOps 团队决定。

故事点

故事点是 DevOps 团队认为要实现特性所需工作量的计量单位。通常我们会使用斐波那契数列来预估。另一种选择是理想时长。重要的是将故事点建立在包含在特性中的故事点的总和上，这比对抽象特征的估计更可靠。

期望的发布火车

一个特性仅当所有下层的故事都实现后才能实现。

申请特性或表明发布计划的用户常常会给这个特性指定一个必须完成的时间。因此，某个特性下的最后一个故事的完成时间必须在这个日期之前。

确定的发布火车

确定的发布火车指搭载一个特性的最后一个故事上线的发布火车，将功能发布给客户。此日期对于确定遗漏的缺陷的原因十分重要。

关联的持续集成应用程序

对于一个特性来说，本质上就是一个变更请求。一个特性的实施过程必须被管理。因此，该特性被链接到一个 CI（配置项），该 CI 标识实现该特性的应用程序。

关联的持续集成服务

除了链接到应用程序，还需要与服务链接确定特性的范围。

单行（one liner）用户故事

一个特性是以用户故事的形式来描述的，这个属性包括这个用户故事。

状态

状态描述了特性位于 DevOps 流程的哪个位置。为状态定义工作流是很重要的。状态突变（Status Mutation）也必须限制在拥有这些权力的特定角色上。从审计的角度来看，这很重要。

审计跟踪

采用某种工具来进行审计跟踪是非常重要的，例如，确定谁创建了、修改了或删除了（逻辑）故事，同时调整了属性（旧的或新的值）。

14.4.3 故事的属性

下面这个清单显示了一个故事的管理涉及哪些方面：

- 特性 ID
- 域
- DevOps 团队
- 变更权威机构
- 基线
- 风险控制
- 故事点
- 状态
- 审计跟踪
- 特性与故事之间的关系

特性 ID

这是指故事被执行所对应的特性。

域

这是执行故事所对应的域，如信息域（功能管理）、应用程序域或基础架构域。一个故事只能属于一个域。

DevOps 团队

一个故事只能由一个 DevOps 团队来执行。

变更权威机构

一个故事的决策机构有多种，如 DevOps 团队自身、产品负责人或变更经理、变更顾问委员会（Change Advisory Board, CAB），或者其他一些组织。

基线

一个故事是一种针对某个对象（信息、应用、基础架构）的改变，在变更中涉及的源文件都属于唯一的基线。

风险控制

已识别风险的管控可以通过各种方式来实现，如变更管理、默认的变更、RFC（变更请求），

或者通过开发—测试—验收—生产流程及其他控制方法。

故事点

每个故事的范围都需要被定义，从而使得这些故事的总和可以被用来决定特性的范围及其优先级（与商业价值结合而定）。

状态

故事状态描述了这个故事位于 DevOps 流程的哪个位置。保持工作流的状态，这一点很重要。状态突变必须分配给拥有这些权力的特定的角色。从审计的角度来说，这一点非常重要。

审计跟踪

采用某种工具来进行审计跟踪是非常重要的，例如，确定谁创建了、修改了或删除了（逻辑）故事，同时调整了属性（旧的或新的值）。

特性与故事之间的关系

特性与故事之间的关系是一对多（1∶n）的关系。

15

敏捷变更管理流程（#14）

15.1 引言

采用 DevOps 的组织正在使用敏捷的工作方式来开发和部署产品及服务。在实践中，变更管理流程仍被使用，但问题是，该如何把变更管理所需的控制和敏捷的工作方式整合在一起呢？

15.2 术语

影响

ITIL®中强大且常用的基本实践之一就是对变更的影响分析。然而，“影响”一词有两个含义。第一个含义是变更的范围，即哪些应用程序和基础架构组件处于本次变更（特性请求）范围内，这个范围也标识了实施这个变更所需的金钱/时间。“影响”一词的第二个含义指一旦没有管理好变更风险，其所造成的后果。对这种影响的计量可以是损失的时间、金钱、声誉、市场份额等。

风险

ITIL®变更管理已经明确了变更的风险基于其发生的概率和影响，可分为高风险、中风险和低风险。

15.3 概念

变更权力机构

ITIL®变更管理流程使用类别对变更进行分级，以及定义所需的变更参与者。该类别通常包括紧急变更、小变更、常规变更和重大变更。小变更、常规变更和重大变更是基于变更的影响（范围）来确定的。变更权力机构已经获得了授权，准许其启动以上 3 种变更和发布已实现的产品。变更权威机构包括开发人员、运维人员、变更经理、变更顾问委员会（Change Advisory Board，CAB）和管理团队（MT）。

15.4 最佳实践

那些认为 DevOps 已经终结了 ITIL®变更管理最佳实践的人，会对一个关于荷兰 10 个组织的管控水平的研究结果感到惊讶[Best 2015a]。这 10 个组织仍然全部在使用变更管理来管控敏捷开发项目，只不过变更管理的角色已经调整了。本文将介绍如何使变更管理变得敏捷起来。

15.4.1 控制层级

这些组织做的第一件事是改变变更参与者的角色。CAB 由新的成员组成，产品负责人现在也出现在 CAB 中。此外，CAB 的授权范围也发生了改变，只负责主题和史诗。若涉及的特性甚至用户故事是高风险的，那么 CAB 仍然会介入其中。CAB 的决策可以快速被实施，因为敏捷项目都已配有愿景和路线图。

垂直分解（参见文章＃12）的特性由产品负责人管理；水平分解的特性必须由多个产品负责人管理，由变更经理负责协调。

15.4.2 浪费

将控制权授予产品负责人确实极大地加快了变更管理流程。由于只有较大的变更才需要由 CAB 达成一致，因此减少了浪费。除了来自发布计划的特性，用户还可以提交自己的特性请求，管理方式取决于需求规模的大小。如果用户提交的是一个史诗，那么就需要由 CAB 审批；如果不是，则只有在这个特性需要由多个 DevOps 团队实现时才需要变更经理介入。但由于产品负责人现在是流程的起点，因此对特性请求可以快速反馈。

是否还能消除更多的浪费呢？答案很简单，是的，当然能。理由是在特性的处理中仍然存在很多浪费，体现在对客户特性请求的分析及功能和故事的管理上（参见文章#11）。

这些浪费存在于特性、验收标准、测试用例、用户故事和关联测试用例的管理中。而且，将特性分解会产生规模、影响和风险各异的用户故事，也是某种浪费。为了消除这些浪费，有必要构建一个更好的视图来展示这些请求的特性的处理情况。

15.4.3 特性管理

一个最佳实践是运用帕累托比率（Pareto ratio）处理特性（需求）来让管理提速。问题是，是否指定应用程序的 80%特性都与 20%的可能变更有关？实际分析表明，情况就是如此。对于一个特定的应用程序而言，必须完成以下工作。

架构图

为应用程序创建一个架构图，其中绘制了程序与环境的接口。应用程序在 CMDB 中被定义为 CI，功能特性则在这个层级进行描述。

组件

细化架构图并绘制其中的组件及相互关系。争取能提取出大约 10 个组件，识别这些组件并用一段话描述它们。

故事

识别出每个组件可能的技术变更，这些变更可能与故事有关。对于构成 80%的特性变更请求的故事，要关注它们的描述。

场景

对于应用程序的一个变更，从客户角度定义可能的问题类型。给它们指定一个场景编号，根据场景描述哪些组件需要调整及需要哪些用户故事。

基本路线图创建了一个固定的模式，所有的特性（请求）可以转换为：

- 场景 ID
- 特性模板

- 应用程序 CI
- 组件 ID
- 故事模板

这种标准化管理的简易性使得模板化成为可能。通过选择一个场景，就可以自动创建上述对象，并在可能的情况下提供对象信息。由于场景已经生成，测试用例的模板也可以链接过来，以便自动生成测试用例。

最后但同样重要的是，每种场景的影响和风险可以提前一次性确认，并事先一次性提请审批。CAB 可以基于这样的变更场景向产品负责人预先给出批准。

16

采用静态需求还是动态需求（#15）

16.1 引言

在使用 DevOps 的组织中，大家感受最深的莫过于终于可以不再被各种框架束缚了。“从 Scrum 中获得乐趣”（Fun with Scrum）如今已成为一种理念，引导大家从 Prince2®，ITIL®（Information Technology Infrastructure Library，信息技术基础架构库），ASL®（Application Service Library，应用服务库）和 BiSL®（Business Information Service Library，商业信息服务库）等框架中解放出来。需求管理就是其中具有代表性的一个方面。在 BiSL 中，需求管理被定义为“明确信息需求”（Specify Information Requirements，SIR）流程，并被完整阐述和规范。但是随着 DevOps 的到来乃至 Business DevOps（DevOps+信息管理）的出现，许多组织已经不再设立需求经理这一角色。本文将阐述这样做是否明智。

16.2 术语

需求

需求是对数据信息、应用程序、基础设施或服务提出的要求。需求可以分为功能性需求和质量需求，质量需求也被称为非功能性需求。

静态需求

静态需求的生命周期会一直延续到产品或服务交付后。服务管理组织会再次使用静态需求来审核产品或服务是否仍然满足需求。本文中后续所提到的“产品”一词也包括服务。

动态需求

动态需求的生命周期随着该需求所链接的产品的交付而结束。

风险

在 ITIL®中，根据变更导致的风险的可能性和影响范围的不同，将变更风险划分为高、中和低 3 个等级。

验收标准

验收标准是针对待交付产品所制定的要求，以确定该产品是否符合事先设定的需求。需求和验收标准的关系是，只有在已知风险得到控制（减轻）或被排除（消除）的情况下，产品才会被发布。要做到这一点，就必须在使用 DevOps 流程实现产品的过程中，采取预防性和响应性的风险应对措施。验收标准中也必须包含对这些措施的有效性评价。

测试用例

测试用例是对如何检查待交付产品是否能正确操作和正常运行的描述。

16.3 概念

基于风险的验收就是将测试重点放到已知风险上，这种风险基于确定的需求。关于需求、风险、验收标准和测试用例、评审及检查清单之间的关系，如图 16-1 所示。

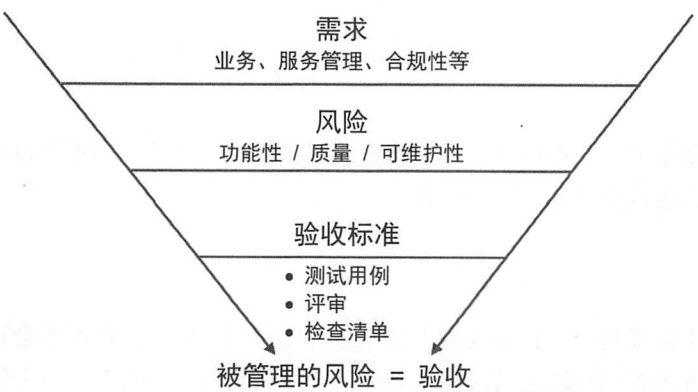


图 16-1 基于风险的验收 [Best 2014a]

需求

需求来自所有的利益相关方并指明了产品需要实现的功能。因此，产品的需求不仅包括用户方需求，还包括项目管理方需求、合规性需求等。

风险

必须对每个需求进行分析，以避免需求未能完全实现所带来的风险。风险可以被分为功能性和质量（非功能性）风险。图 16-1 也为我们展示了可维护性风险，这类风险与服务方有关，与用户无关。虽然可维护性也属于质量的一部分，但因为它太重要了，所以在图 16-1 中将其从质量管理中单独分列出来。

需要强调的是，除了基于风险的验收，还可以通过使用基于 DevOps 流程中的制品来进行风险分析，以识别风险。这些风险需要对照需求进行测试。由于还没有对这些方面加以考虑，采用这种方法通常会导致需求增加。

验收标准

验收标准是针对待交付产品所制定的要求，以确定该产品是否符合事先所设定的需求。需求和验收标准的关系是，只有在已知风险得到控制（减轻）或被排除（消除）的情况下，产品才会被发布。要做到这一点，就必须在使用 DevOps 流程生产产品的过程中，采取预防性和应急性的风险应对措施。验收标准中也必须包含对这些措施的有效性评价。

测试用例

验收标准的测试需要进行大量测试和/或评审和/或检查。这是测试管理的范畴。然而还有许多测试类型与验收标准并无关联。

16.4 最佳实践

本文讨论的是静态需求与动态需求之间的区别。如图 16-1 所示，先有需求后有风险管理。在文章#9 中也介绍了产品特性与需求之间的关系。因此，可以把需求看作一个被管理的对象，各种属性被附加到这个对象上，如特性/场景、测试用例、声明的对象等。

16.4.1 动态需求和静态需求

采用动态需求还是静态需求的核心问题是，在交付产品后还需要管理哪些信息来维护产品。遗憾的是该问题的答案并不是非黑即白那么简单。静态需求适合的产品满足以下一项或几项条件：

- 该产品属于核心业务。
- 该产品受到合规性检查限制。
- 该产品经常被修改。
- 该产品存在大量已知风险。
- 该产品对相关的知识积累要求非常高。

相对地，动态需求所适合的产品满足以下几项条件：

- 该产品并非核心业务，即使功能失效，也不会对公司产生较大影响。
- 该产品不需要满足法律要求。
- 该产品不需要经常修改。
- 该产品不存在或只存在极少风险。
- 该产品不需要特定的知识和技能，并且容易快速学习和了解。

此外，符合静态需求条件的产品也可能具有动态需求。例如，添加一个临时需求或删除一个不再需要维护的需求。

16.4.2 一些思考

正如文章#9 所述，可追溯的需求是部署流水线非常重要的一个质量要求。不完全符合规范的需求可能带来某些小便利，但同时会带来更多的麻烦和隐患。

在某个时刻我们必须对需求是静态的还是动态的做出判断。如果需求是动态的，那么这个需求就会有部分无法追溯。

问题在于这样做实际上能为我们节省多少时间呢？一旦各种工具被整合且运行良好，时间的花费将不再是个问题。动态需求也需要进行测试，最好是自动化测试。为什么要把那些仍然有用的东西摒弃掉呢？只有在所有的需求都以文档形式编写且与其他系统没有接口对接的情况下，采用动态验收标准才有意义。这种情况下需求往往无法被追溯。

所以，需求的创建和编写非常重要。因此 BiSL 中关于“明确信息需求”（Specify Information Requirements）流程的最佳实践，对 DevOps 的也同样意义重大。此外，基于 TMap NEXT 的最佳实践进行的需求测试在 DevOps 中也非常重要。最后，ITIL®中把验收标准作为一种控制机制的最佳实践，不应该被 DevOps 排除和忽略。

17

软件配置项（#16）

17.1 引言

在 ITIL®的第一个版本中，流程配置管理（Process Configuration Management，PCM）就已经被定义了。在 2011 年的最新版本中，该过程的名称已更改为服务与资产配置管理（Service & Asset Configuration Management，SACM）。然而，在软件开发领域，还有一个配置管理过程，常被命名为软件配置管理（Software Configuration Management，SCM）。本文介绍这两个过程以及它们之间的关系，并且关注软件配置项（S-CI）的使用。

17.2 术语

服务与资产配置管理

这一过程的目的是界定和维护与 ICT（信息通信技术）服务相互关联的信息和 ICT 基础设施的逻辑模型，以及提供那些 ICT 服务所需的组件。

配置项

配置项（CI）是对服务组织所管理的对象（服务或产品）或对象组件的注册。CI 具有多种属性用以描述对象或组件。配置项定义的对象可以是服务、软件、硬件、通信工具、文档和程序。注意，配置项只表示对象的注册，而不是对象本身。

配置管理数据库

配置管理数据库（CMDB）是一个定义了 CI 和 CI 关系的数据库。在设计 CMDB 时，必须考虑以下事项：

- 范围。意指在 CMDB 中描述了哪些对象。
- 深度。意指有多少“父子关系”被识别。
- 详细程度。意指配置项应该定义多少个属性。

这 3 个维度是通过 SLA 直接或间接确定的。在 SLA 中，定义了产品和/或服务。对 CMDB 的一个常见的误解是 CMDB 包含这些物理对象，然而，CMDB 只是一个包含以 CI 形式存在的对象数据的数据库。

CMDB 基线

CMDB 基线是特定版本的一组配置项（CI），它们一起形成稳定的服务和/或产品。如在主要版本发布后，所有相关配置项都可以定义为基线。如果后续发生干扰，可以查看自上次基线以来所做的更改。如果生产环境变得不稳定，基线也可被用于回归上一个基线。

软件配置管理

在软件工程中，软件配置管理（SCM）跟踪和检查软件的变化，是配置管理更大的跨学科领域的一部分。软件配置管理的实践包括审计控制和基线的建立。如果出现问题，软件配置管理可以确定发生了什么变化以及谁改变了它。如果配置运行良好，软件配置管理可以确定如何在多个主机上复制它[维基百科]。

软件配置项

软件配置项（S-CI）是对由 DevOps 开发过程所管理的对象的描述。软件配置项分配了多个属性用于描述对象或组件。一个软件配置项可包括一个第三代或第四代语言的元数据、一个 SQL 脚本、一个数据库结构和应用程序所需的任何其他对象。

软件配置管理数据库

软件配置管理数据库（S-CMDB）是定义软件配置项及其关系的数据库。基线是软件配置管理数据库的一部分。与 CMDB 不同，S-CMDB 是一个源代码和源代码软件配置项（作为元数据）的存储库。诸如 GIT 和 SubVersion（SVN）等工具为用户提供正在开发的软件的集成镜像，并根据元数据和文件夹结构控制该软件的配置。

S-CMDB 基线

S-CMDB 基线是在某个时间点对产品属性已达成一致的描述，这个描述被视为定义变更的基础。变更是从这个基线状态到下一个状态的运动。识别重要的变更是基线识别的核心目标[维基百科]。

17.3 概念

CMDB 的使用

数十年来，运维一直都在使用 CMDB。ITIL®的所有过程都使用 CMDB。比如 CI 与事件（incident）连接，用以标识事件的位置。变更请求也由一个或多个配置项表示。与此同时，SLA 标准与配置项相关联，因此，服务台知道解决该事件需要花费的最长时间是多少。以此得出，CMDB 是所要管理的对象的核心定义。

S-CMDB 的使用

S-CMDB 概念的提出可能比 CMDB 更早，它的使用与 CMDB 不同，因此，它描述的对象比 CMDB 描述的小得多。除了 S-CMDB，S-CI 的版本管理也至关重要。为了编制新的基线，只能使用被批准的 S-CI 版本（见图 17-1）。

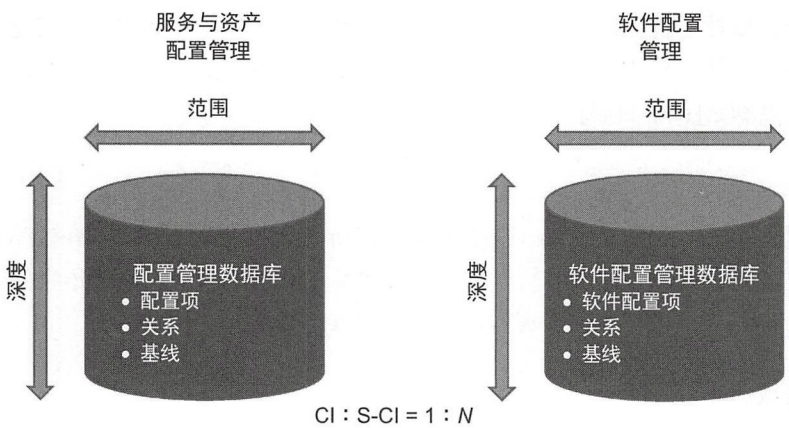


图 17-1 CMDB 与 S-CMDB 的比较

17.4 最佳实践

S-CMDB 和 CMDB 都属于 Dev 和 Ops 的基础管理。由于在管理上它们通常没有被关联起来，因此很难从源代码的角度发现事件的本源。此外，重要的是要认识到 DevOps 中 S-CMDB 的功能，以便能正确地实施。

17.4.1 CMDB 和 S-CMDB 的关系

在使用 CMDB 和 S-CMDB 之初，通常会提出的第一个问题是如何将这两者联系起来。通常，应用程序的一个部署是通过创建一个或多个 CI 和/或注册一个或多个 CI 的新版本来完成在 CMDB 中注册的。

就 S-CMDB 而言，一个新的发布需要定义新版本的基线。基线被用于通过构建服务器创立一个构建（Build），每个构建获得一个唯一的编号。

建立 CMDB 和 S-CMDB 之间的关系非常重要。例如，通过在 CMDB 的配置项中捕获应用程序的构建号（Build ID）。作为结果，Ops 推出的产品和 Dev 开发的产品之间形成了可追溯性。

图 17-2 展示了 CMDB 和 S-CMDB 之间的关系。通过特征/故事关系可以看出二者的关系。

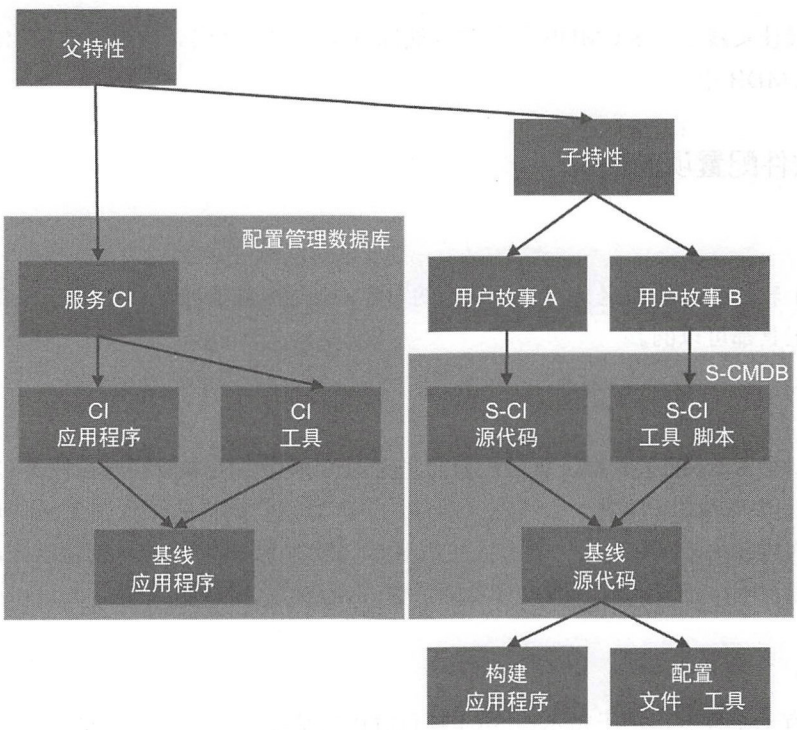


图 17-2 CI 和 S-CI 的关系

17.4.2 S-CMDB 的使用

S-CMDB 确保 DevOps 团队提供以下功能：

- 对象定义：应用程序运行所需的软件配置项定义。因为软件配置项也具有基础设施配置文件（基础设施即代码）。
- 版本管理：防止版本被覆盖或丢失。此外，版本管理可以让更多的 DevOps 成员通力合作，实现目标。
- 基线：提供一个软件配置项标记，用于定义属于一起并可由构建服务器转换为目标代码的软件配置项的基线。
- 连续性：S-CMDB 可以返回到以前的版本。

通常，软件配置管理会被分配一个更大的角色，包括部署。

然而，最佳实践是让 S-CMDB 只包含存储库意义上的源代码。构建过程输出到制品库中，而不是到 S-CMDB 中。

17.4.3 软件配置项的使用

对象定义

任何一个软件配置项被定义为一个唯一的对象。一些服务管理工具允许将 S-CI 与 CI 关联，但不是所有工具都可以的。

版本管理

要使用一个 S-CI 的某版本，软件配置管理工具必须提供版本控制。各种 S-CMDB 架构有各自独特的功能。版本管理的一个基本原则是，多人可以在同样的源代码上同时开展工作，而不会丢失源代码。此外，有必要一旦有人更改 S-CI，就有新版本的 S-CI 创建出来，这样元数据的管理就会很清晰：谁签入，哪些内容发生了更改，是什么时候更改的。

基线

S-CI 具有一个标签，用于指示该 S-CI 打包时用的基线。

连续性

S-CMDB 的一个重要特性是可以基于源文件重新创建应用程序。这是非常必要的，因为备份并不总能提供解决方案。

18

版本控制（#17）

18.1 引言

版本管理是 DevOps 部署流水线的一个重要方面。许多 DevOps 团队很晚才认识到每个对象都需要有一个（正确的）版本。本文给出了要进行版本控制的对象列表，并解释了为什么要版本化这些对象。

18.2 术语

对象

在本文中，对象被定义为通过部署流水线部署的所有东西，如软件产品、软件配置文件、中间件、中间件配置文件、硬件配置文件等。此外，为了管理部署流水线而创建的所有对象都被视为对象，如主题、史诗、特性、故事、需求、测试用例、CI、S-CI 和文档等。

版本

软件版本化是将软件开发和发布时的唯一状态分类的一种方法。

版本号

没有通用的版本号逻辑。 但是，最常用的一种是：

版本（W.X.Y.Z）

- W = 主版本号（重大升级，不同主版本的库之间是不兼容的。——译者注）

- X = 次版本号（增量升级，增加一些新的接口但保留原有接口。高版本号的库向后兼容低版本的库。——译者注）
- Y = 维护版本号（仅修复 Bug）（修改错误、改进性能等，不添加新接口，也不更改接口。在主版本和次版本相同的前提下，不同维护版本之间完全兼容。——译者注）
- Z = 构建号（或源代码管理版本号）

版本控制系统

[维基百科]版本控制系统（VCS）是一个或一组计算机程序，可用于管理文档、程序或存储在计算机文件中的其他信息的变更。VCS 最常用于软件开发管理，以便实现多人同时对相同的文件进行更改。变更通常由代码、修订号或修订级别来表示。每个更改都会与时间戳、进行更改的人关联起来。更改可以进行比较、恢复和选择合并。版本管理系统通常是一个单独的程序，但它也会内置于某些软件中，如文字处理器（如 Microsoft Word，OpenOffice.org Writer，KOffice，Pages，Google Docs）、电子表格（如 OpenOffice.org Calc，Google Spreadsheets，Microsoft Excel）以及不同的 CMS（内容管理系统）。版本管理也是维基软件的一个特性，做出的变更总是可以被逆转的，如蓄意的破坏。

18.3 概念

版本管理

版本管理是管理对象变更的过程。

版本策略

版本管理重要的是一开始就确定版本策略，这个策略是关于如何做出重要的选择的。这些选择可以通过回答以下问题来完成：

- 哪些对象需要版本化？
- 更新版本的原因是什么？
- 使用哪个版本号格式？
- 版本号如何更新？
- 身份管理如何与版本控制相关起来？

可以基于版本策略选择正确的工具。

18.4 最佳实践

本文回答 3 个问题：第一个问题关于版本管理的商业论证；第二个问题关于哪些对象进行版本化以及在 DevOps 流程中的何处发生；最后一个问题关于为什么对这些对象进行版本管理很重要。

18.4.1 什么是版本管理的商业论证

在 DevOps 的部署流水线中，每个对象都必须有一个标识（ID）和版本。 版本管理的商业论证（Business Case）假定存在以下风险：

- 不同的人会相互覆盖针对同一个对象所做的变化。
- 丢失了一个月的程序开发成果。
- 无法确定缺陷是何时开始产生的。
- 不知道哪个版本的对象在哪个环境中。
- 不知道是谁及什么时候改变了对象。
- 不知道哪些对象的哪些版本可以一起工作。

为了消除这些风险，版本管理帮助 DevOps 团队做到：

- 通过以下方式允许更多的人在同一个产品上一起工作：
  - 签入/签出机制。
  - 用于创建分支版本的工具。
- 识别对象，以便能够把对象恢复到旧版本。
  - 存储这些版本的旧版本和元数据，如更改的内容、是谁更改的、何时更改的、何时批准的。
- 通过追溯缺陷产生的源头版本来加速缺陷的解决。
  - 一旦发现缺陷就可以追溯到产生缺陷的编程版本。
- 识别对象。
  - 编号的版本能够识别组件对象。
- 可以将需要一起构建、测试和部署的版本化 S-CI 定义为一个基线版本。

18.4.2 对象的版本化是在 DevOps 流程中哪里发生的

表 18-1 给出了要版本化的对象的概述，这些对象按照 DevOps 的阶段进行归类。

表 18-1 每个 DevOps 阶段需要版本化的对象

阶 段	版本化的对象	工具化
规划	主题、史诗、特性	需求管理工具
编码	S-CI、基线、数据模型	软件管理工具 数据建模工具
构建	构建 ID	构建服务器工具
测试	测试数据、测试用例、测试集、测试运行	测试管理 测试自动化工具
发布	发布说明、部署脚本、配置文件	发布工具
运维	数据文件、协议	数据文件管理工具
监控	监控配置文件	监控系统

18.4.3 为什么这些对象需要版本控制

表 18-2 给出了与 DevOps 各阶段相关的对象的概述。

表 18-2 版本控制的对象以及版本化的原因

对 象	版本化的好处
主题、史诗、特性	主题、史诗和特性是需求的占位符。这就意味着只要改变其中一个对象，那么包含在部署流水线中所有与之相关的对象都应该被改变（见文章＃09）
S-CI（软件配置项）	持续集成要求高频率的检出和检入。 每次 VCS 应该生成一个版本，用来跟踪对象的变化。这意味着如果有源代码丢失或无法解决的严重错误，前一个版本（能够恢复到历史的任何版本。——译者注）仍然可用
数据模型	应用程序或报告中的信息必须进行良好的定义。应用程序接收、转换和发送/导出的信息随着时间而发生变化，因此，数据模型的版本控制是数据质量管理最重要的方面
基线	基线由特定版本的 S-CI 组成。DevOps 团队通过测试用例来检查所有这些 S-CI 是否能够一起正常工作。但是，如果在测试之后使用了错误的 S-CI 版本，那么在后续的部署流水线中将会进一步发生错误。所以，VCS 应该跟踪 S-CI 和版本对基线的变化
构建 ID	编译的对象通常是不进行版本控制的，但它们是通过使用独特版本的软件创建的，所以，构建 ID 应该指向所使用的基线版本
测试用例	测试用例用于检查是否满足需求。测试之前必须知道测试对象，一旦需求或对象形成新版本，测试用例就可能失效，就需要进行调整，因此，测试用例也必须有一个版本。当然，在回归测试的情况下，如果测试用例总是在同一个测试集中使用，则版本化可以在测试集

续表

对 象	版本控制的好处
测试用例	级别进行管理
测试数据	测试数据供测试用例使用。通常情况下，测试用例会创建和删除自己的测试数据，但大部分时候也需要一套标准的测试数据集，并且需要进行版本控制，以便检查使用它的所有测试用例是否能够正常工作
测试运行	测试运行使用特定版本的测试数据集，所有发现的缺陷应该指向使用的测试数据集版本
配置文件	不仅是软件，其他所有对象如数据库管理系统、操作系统、防火墙等都需要配置设置。设置包括路径、数据库名称、消息队列等。每个环境可能需要不同的设置，所以对象和配置文件需要分开管理。在对象部署之后，要使用正确的配置文件的正确版本来配置对象，因此，配置文件的版本化对于保持部署流水线正确运行非常重要
部署脚本	部署一个完整的环境需要很多部署脚本，因此需要对这些脚本进行版本化，以便将它们与配置文件的版本进行匹配
发布说明	发布说明针对每个版本进行更新。与发布的应用程序的版本保持同步是非常重要的
数据文件	许多应用程序会产生输出文件给其他应用程序进行事务处理，对这些输出文件的格式进行版本化是非常重要的。另外，创建文件的应用程序的版本是解决事件的重要信息
协议	应用程序之间的通信是基于协议进行的，这些协议需要进行版本管理。许多应用程序也会检查由其他应用程序使用的协议版本，以便就通信达成一致
监控配置文件	监控配置必须正确才能检测事件，通过使用某个版本化的监控配置可以得到很好的保障

19

标准、规则和指南（#18）

19.1 引言

生成高质量的代码是敏捷工作的基本原则之一，精益也同理。精益使用一个原则：如果有错误发生，生产线上的产品就不能再被继续向下传递。问题是如何在 DevOps 方法中应用这一原则。

19.2 术语

注释

注释是对一段源代码的解释。

分支

分支是使用源代码的副本来创建两个分别开发的版本的一种技术。在这些分支被开发者改变之后，会通过合并进行重组。分支有多种形式，因此，DevOps 团队必须选择适合的分支形式，这种选择也被称为分支策略。

就绪的定义

就绪的定义（DOR）是敏捷 Scrum 中使用的关卡，用于确保开发团队只处理那些符合 DOR 要求的特性。DOR 由开发团队拥有。

完成的定义

完成的定义（DOD）是敏捷 Scrum 中使用的关卡，用于确保在 DOD 测试发生之前，产品负责人不接受冲刺中的任何产品。DOD 由开发团队拥有。

事态目录

事态目录是应用程序的所有事态的列表，包括标识和类别（信息、警告或异常）。

事态管理

这是 ITIL®的一个流程，用于定义哪些事态值得识别和捕捉，可能是信息事态，如启动或停止进程；也可能是异常的各种可能性，被称为警告。磁盘 80%已满就是一个例子。如果应用程序或基础设施组件无法正常运行，就会出现异常。

标准、规则和指南（SRG）

标准是一个必须始终符合的约定（政策），没有例外。如果不符合标准，就是一个缺陷。规则是在特定条件下可能有所不同的标准，偏差必须加以记录并论证。指南是一个非强制性的建议，但是根据此建议采取行动可以做出更好的可维护的产品。

19.3 概念

质量控制与保证

质量保证（QA）是将完成的产品与质量控制（QC）进行比较的概念。无论如何，质量控制必须具体，因此，必须就工作质量达成协议。SRG 是具体的 QC，可以应用于像 DOR 和 DOD 这样的关卡。QC 和 QA 并用（QCA）是获得高质量代码的手段。

19.4 最佳实践

人们无法质疑口味，这是个人的事。这也适用于源代码的质量。源代码的质量只有在定义了 QC 的情况下才能被测量。在 DevOps 中，敏捷是以敏捷 Scrum 的具体方式来实现的，其中某些质量控制被定义为 DOR 和 DOD。但是，敏捷 Scrum 并不定义 QC 的内容，QC 的内容必须由 DevOps 团队来确定。尽管如此，还是有一些普遍的方法，而且每个人都能从中受益。因

此，本文以 SRG 的形式给出了一些 QC 的例子。

19.4.1 模块化

一些开发人员声称，当在 DevOps 团队中工作时，不知道开发何时停止。因此，如果你创建了应用程序的第一个版本，一个需要面对的选择是，以可扩展的方式来开发它，还是选择更简单的结构。当定义了以后将支持软件模块化的独立类后，软件可以很容易地扩展。不过这会比采用简单的结构要花费更多的时间。

然而，问题在于，与从一开始就采用模块化结构相比，之后的重构会需要（浪费）多少时间？ 一个好的解决方案是允许基于产品市场发布时间而偏离模块化，但前提是产品负责人必须已经在产品待办事项上规划了优先于进一步开发的重新设计工作。因此，在继续工作之前，有必要为偏离（模块化）付出时间和金钱以重新设计。在这种情况下，产品负责人不再决定该重新设计的优先级，所以模块化是一个规则。

19.4.2 分支

另一种 QC 是决定是否使用分支的方法。一旦有分支，就会有浪费。毕竟，合并必须执行并且不会产生商业价值。除非绝对必要，最好不要制作软件的一个分支。因此分支是一个可以偏离的规则，但是这种偏离必须有充分的论据。有些组织根本就不使用分支。

19.4.3 版本

在 DevOps 中，默认应用程序生成的任何对象都会被提供一个版本号，并且被包含在存储库中。对于源代码来说，这种情况发生得越来越多，几十年来一直如此。另外，不包含源代码的对象是 DevOps 团队在周期中生成的对象，如数据库模式、更改防火墙设置的脚本、安装脚本等。因此，在这种情况下，DOD 必须是一项 QC，所有需要应用于其他环境的对象所做的所有更改都必须签名。该脚本必须有版本号，并且与源代码一起包含在同一个存储库中。

19.4.4 代码质量

QC 的另一个例子是软件代码的质量，如函数名称、函数大小、库的使用等，这些是编译器不检查的源代码的方面。例如，一个编译器不会给出一个关于注释度太低的错误信息。函数的大小也是一个选择。最佳实践之一是一个函数的源代码行最多只占一页 A4 纸。论据是，该功能易于被理解、调整和测试。此外，签入可以每天进行几次，以检查该函数是否正常工作。

很多编译器无法控制的代码质量问题可以通过如 Tiobe 的代码扫描器记录。该工具甚至给出了代码的质量标签，其中“A”是最高质量，“F”是最低质量。因此，Tiobe 会指出哪个源代码在质量方面不达标。还有以像素为单位的整体代码质量的图形概览（见图 19-1）。如果屏幕上有一个红色像素可见，点击这个像素就可以看见它所代表的代码行。相应的解释包括编写代码行的程序员的名字和检测到的偏差。质量标签可以作为一个 DOD 的标准。

19.4.5 事态管理

DevOps 团队的优势在于开发人员和管理员都在同一个团队中。这并不意味着开发人员不必再担心事态管理。事实上，管理员（与开发人员同在一个团队）要指导开发人员处理应用程序中的事态。有许多 SRG 是与事态管理相关的。

标准的一个例子是，应用程序的所有事态都列在事态列表中。因此，DOD 应该提供一个条目，来验收是否与每个已实现的功能相对应的事态目录已被更新。之所以将其作为标准，是因为事态应该被自动监控，而且最好被自动处理。

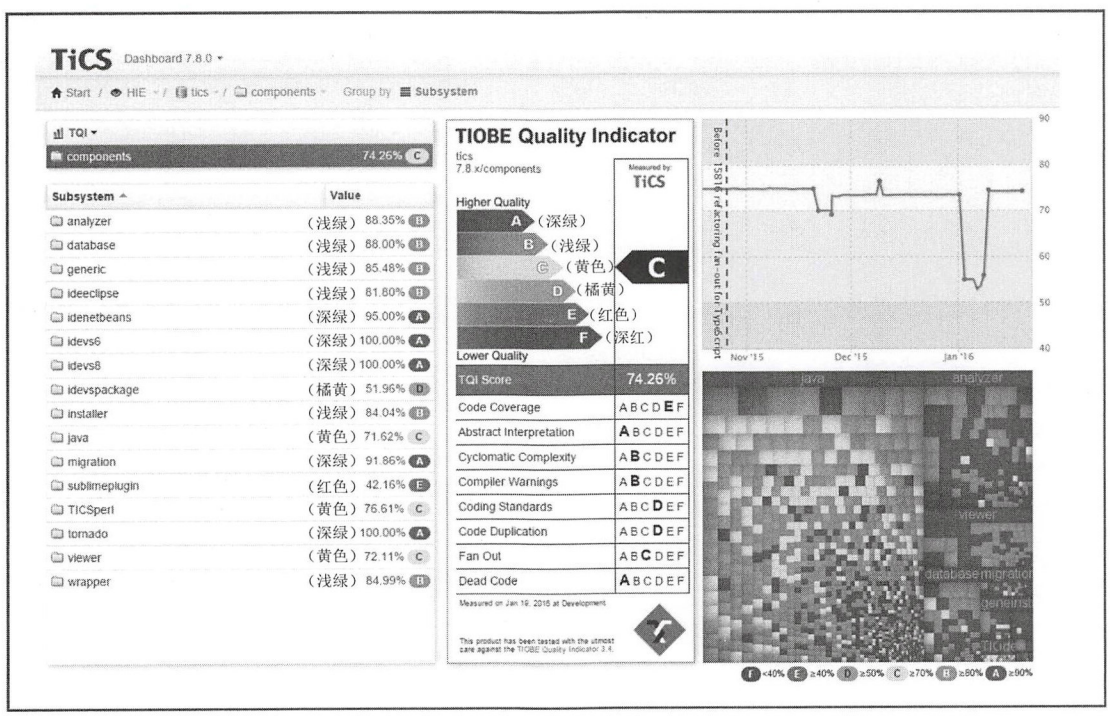


图 19-1 TiCS 仪表板

19.4.6 结论

使用 DevOps 意味着可以创建高质量的源代码。使用 SRG 进行具体的质量控制，可以由 QA 在 DevOps 流程的关卡检查。一致性在 SRG 中是非常重要的。工具可以加速 QA 检查。

20

分支模式（#19）

20.1 引言

代码分支是编程的基本选择之一。本文介绍了在 DevOps 范畴内有关分支的基础知识。

20.2 术语

分支

分支是一种通过复制源代码生成两个版本并进行分别开发的技术。开发者在修改完这些分支后，通过合并分支进行代码重组。由于分支形式多种多样，因此，DevOps 团队必须选择适合的分支形式，这种选择也被称为分支策略。

抽象分支

通常情况下，旧的软件架构（单体架构）无法满足通过部署流水线进行维护的要求。这种单体架构的软件不是模块化的，因此在发布时，必须作为整体进行升级。抽象分支方法为识别单体应用中较大的软件模块提供了可能。 通过隔离这些模块并定义一个健壮（strong）的接口，使得冻结模块并完全重写（重构）它成为可能。如果新开发的软件代码通过了测试，就可以将代码所在的分支合并到主干中。

合并分支

合并分支与创建分支正好相反。在合并分支的过程中，需要识别相同的 S -CI（软件配置项）两个版本间的冲突，并且通过消除冲突项的方式进行解决。

发布分支

发布分支是分支的一种特殊形式。它是经过部署流水线的一个最新基线（主干）的副本，而主干上的开发仍在继续。如果发现缺陷，可能不会修改发布分支，但是必须在主干中进行修复。如果需要，则在发布分支中通过更新补丁进行修复。不过，只有一个版本的 S-CI 将被更新。

特性分支

特性分支法意味着每个用户故事或特性都是在单独的分支上进行开发的。只有测试人员确认用户故事或功能测试通过后，分支才被合并到主干中，以确保主干处在一直可发布的状态。

软件配置项（S-CI）

软件配置项（S-CI）是对 DevOps 开发过程中被管理对象的描述。软件配置项使用多种属性来描述对象或组件。软件配置项可能包括第三代或第四代语言的元数据、SQL 脚本、数据库模式或其他任何构成应用程序所需要的对象。

20.3 概念

分支策略

分支策略是指对源代码进行分支管理所制定的相关处理策略。

前向发布

前向发布是一个概念，是指不会在 DTAP 各个阶段（开发—测试—验收—生产）中创建任何分支（仅限发布分支），并且也不恢复旧版本。同时，如果在 T-A-P 环境发现缺陷，补丁代码的部署发布也必须通过部署流水线进行。

精益

精益原则来源于日本的制造业。该词由 John Krafcik 在 1988 年发表的《精益生产系统的胜利》（*Triumph of the Lean Production System*）一文中首次提出。对于很多人来说，精益就是帮助识别和不断消除浪费的一套工具。随着浪费的消除，既提升了质量，同时还减少了生产时间和成本。

这套工具中有非常重要的 SMED（一分钟换模法）、价值流映射、5S 法、看板（拉式系统）、poka-yoke（防呆）、全面生产维护、消除时间批处理、混合模型处理、等级顺序聚类、单点调度、重新设计工作单元、多进程处理和控制图（检查不均匀）。[WIKI]

20.4 最佳实践

源代码采用分支方式进行管理的潜在问题是 S-CI 的两个版本难以合并。随着时间的推移，这种风险会持续增加。现在有很多工具能够发现与识别那些彼此不会产生冲突的调整内容，从而可以非常智能地执行合并，但是这些工具无法处理类似函数重命名这样的语义变化的情况。如果特性分支的生命周期很短，那么执行分支合并时通常不会存在大的问题，但是从本质上讲，这也是浪费，因为分支合并活动需要花费时间，并且容易出错。

如果不创建分支，则不需要合并，当然也就不存在版本管理的问题了。

20.4.1 分支适合 DevOps 吗

关于分支为什么不适合 DevOps 存在一些争议：

- 因为分支会产生浪费，所以不符合精益原则。分支会产生额外的任务，即合并。另外，合并过程中的冲突识别本身也是一种浪费。
- 不符合持续集成的原则。代码必须经常合并到主干中，同时，代码需要一直处于可执行（可发布）的状态。
- 分支会导致代码不再具有整体性，因为代码实际上是被部分冻结的。
- 分支会使重构变得困难。
- 自动化合并工具无法解决所有合并的冲突，如语义冲突（重命名对象）。

20.4.2 事件

不采用分支的代价是，如果在生产环境中发生事件（incident），则没有快速的解决方案。最快的解决方案就是部署流水线。当然，可以通过在 DTAP 流程涉及的相关环境中部署补丁并进行回归测试来验证补丁的有效性，从而提升事件解决的速度。实际上，部署流水线被停止了。在允许这样做的组织中，必须经常提供高级经理的签名，以控制这些部署流水线关闭的频率。

20.4.3 前向发布

前向发布的思路是，任何 S-CI 在一个环境中检出，并且其部署没有出错，将不再被恢复。原因是很多 DevOps 团队使用相同的部署流水线。一旦已经检出的 S-CI 被重置，则所有其他依赖其的 S-CI 也必须重置。这种干扰可能是无关紧要的，其余的测试用例也可以运行得很好。因此，必须等待通过常规部署流水线的缺陷补丁。前向发布可以防止使用分支。

20.4.4 分支策略

每个 DevOps 团队都有自身的成熟度，而每个应用程序也有自身的特点，因此，必须选择一个分支策略。理想的分支策略是不使用任何分支，并且通过快速部署流水线来执行快速修复的操作。如果 DevOps 成熟度还不高，那么也可以暂时选择发布分支或特性分支策略。

20.4.5 结论

DevOps 包含持续集成、持续交付和精益方法，这意味着分支与 DevOps 的理念是对立的。最佳方式是不使用分支，而使用完善的持续集成（CI）和高度自动化的部署流水线来实现可靠的持续交付（CD）。

21

异常管理（#20）

21.1 引言

从操作的角度来看，异常管理是应用程序或系统软件组件的最重要的功能之一。

这种设计的方式决定了受控对象在功能和质量方面纠正偏差的能力。

21.2 术语

事态目录

事态目录是应用程序的所有事态的列表，包括标识和类别（信息、警告或异常）。

事态管理

这是 ITIL®的一个流程，它定义了那些有价值的事态被识别和捕获的过程。它们可能是正常事态，如启动或停止进程；一些可能的异常事态，被称为警告，如磁盘空间已经占用 80%了。

在有些情况下，如果应用程序或基础设施组件不能正常运行，则称为异常。

异常

异常是应用程序或系统软件正常运行被扰乱的情况。

21.3 概念

异常管理

这是一个流程，用于确定可能的干扰，在软件中检测异常并进行处理，以达到纠正的目的。

事件管理

异常（exception）和事件（incident）之间的区别是，异常是一个可能导致事件的事态（event），但其本身不是一个事件。事件是偏离服务级别协议（SLA）的可能偏差。许多监视工具提供了基于异常自动创建事件的能力。然而，在某些情况下，多个异常事态实际上来自同一个事件。在这种情况下，可以创建一个事件并将多个异常事态与此事件关联。

健康模型

健康模型是一种概念，用来在实现主题、史诗或特性前确定可能的错误路径（error paths）。错误路径不可能被穷举，能确定 80%就非常不错了。在软件的生命周期中，健康模型可被不断地扩充。

除了确定可能发生的干扰，健康模型分析可以在意外事态发生后找到那些可用信息。更重要的是，确定该情况目前是否还在反复发生。最后，健康模型指出哪个监控设备具备检测该意外事态的能力。

21.4 最佳实践

在实践中，开发人员只关注信息系统（应用程序、系统软硬件）是否按照快乐路径（happy path）运行。事先并没有或极少对替代路径（alternative path）和错误路径进行调查。替代路径是快乐路径之外的另一种可能的处理或交易方式，但替代路径并非有什么错误。而错误路径是不正确的，这种错误必须予以考虑。

本文介绍了 DevOps 的视角下的异常管理过程，以提高客户服务质量。

21.4.1 哪里会发生异常

将异常管理交给开发人员是很常见的，他通常会分析在源代码流中是否可能出现不同的行为或异常。

通常是在离开代码上下文时发生的，比如：

- 调用内部或外部应用程序功能。
- 使用数据库管理系统的 API 函数。
- 基于功能将信息写入存储介质。
- 基于功能与基础设施的通信服务（打印服务、电子邮件服务、LDAP 服务等）。

开发人员在该程序里取回返回状态。如果返回状态不是“0”的话，那么就发生了异常。

21.4.2 运维人员需要哪些信息

许多程序员不知道运维人员需要什么信息来修复中断问题。

错误消息处理的错误例子有：

- 经过一夜 40 万个交易的批处理后，应用程序日志文件中出现了一个错误，没有人知道该怎么办。
- 在一个每年有数千万条信息的全国性系统中，当出现错误消息时，一个应用程序在计算机中心的控制台给出了一个 Java 栈跟踪的 Dump。开发人员假设他会在中断发生时被呼叫，那么这些信息会传递给他。
- 一个应用程序在错误日志中写了 4 行文本，然而监控设备只处理每个错误的第一行，另 3 行会被忽略。
- 错误消息中的错误代码不是唯一的。
- 一个错误消息是这样写的：“出错了，如果频繁发生此错误你需要联系系统管理员。”
- 没有错误消息，但显示一个系统页面。
- 没有错误消息，因为程序中没有写异常代码。
- ……

通过 DevOps，开发人员和运维人员合作更紧密，这样他们能够互相学习如何编写更好的异常处理程序。下面的信息对于异常管理 SRG 非常有用：

（1）每个错误消息都有唯一的编号。

（2）错误消息是动态的，这意味着在软件里文本没有被硬编码。但如果从异常文件或异常数据库检索，这意味着：

- 能够被维护。
- 能够被转换。
- 通过事态目录提取。
- 能提供元数据。

（3）错误消息不仅发送到控制台和/或用户的屏幕，而且发送到日志文件。

（4）错误消息通过配置元数据来提供（如果可以的话）：

- 产生异常的 S-CI（应用程序）。
- 异常编写的位置 （函数名和代码）。
- 尝试与其通信的 S-CI 或 CI。
- 异常发生时托管本 S-CI 的 S-CI 或 CI 。

（5）错误消息能够提供以下运行状态（run state）元数据（如果可以的话）：

- 异常的日期和时间。
- 有关业务交易的关键值。
- 用户 ID 或系统账户。

（6）错误消息提供了以下解决信息：

- 类型。有异常（exception）、警告（warning）和信息（information）这 3 种。
- 严重程度。每个代码对应一个事件：
  - 000 = 状态信息报告
  - 010 = 连接错误
  - 011 = 死锁错误
  - 012 = 授权错误
  - 999 = 数据损坏错误
- 解决人。有用户、功能管理员、应用程序管理员、基础设施管理员。
- 错误消息同时提供了解决方案：
  - 通过（×××方法）重启系统
  - 通过（×××方法）恢复备份

— 通过（×××方法）分析基础设施的不可用性

建议 1 为错误消息提供唯一的编号

通常提供一个通用的数字。为错误消息提供唯一的属性编号，对持续监测是十分重要的。在开发中，单元测试将帮助验证错误消息是否被监控，正确的信息是否写入了日志。每个异常都必须在开发环境中被监测到，因此，一个唯一的错误状态是很重要的。在其余的 DTAP 中，快速识别问题也十分重要。因为只有信息具有唯一性，才能够做到快速识别。

建议 2 动态消息

重新编译源代码来调整错误消息显然是一种浪费。外部捕捉（而非硬编码）也使得应用程序的多语言的配置成为可能。很多应用系统不知道什么会出错，因为错误消息不符合唯一性的条件，并且是被硬编码的。更糟糕的情况是根本就没有异常处理程序。

建议 3 保存错误消息

如果用户电话报告一个事件，并给出对应的错误代码，那么相同的信息在日志文件中也被记录是很有用的。这是因为即便用户打了电话也无法确切知道在错误消息发生前，他在系统里执行了哪些步骤。

建议 4 配置数据

配置信息对于确定错误消息的位置及问题管理（problem management）都非常重要。这看上去有很大的工作量，但是可以通过一个集中异常处理器来编写，只需要花费非常少的额外时间。

建议 5 运行信息

从安全的角度来看，并非所有的信息都可以包含在日志文件中，但键值（key values）一般是允许的。当然，在事务链（transaction chains）中，这是非常有用的信息。要匹配监控系统中的事态与服务台的事件，用户 ID 非常重要。

建议 6 解决信息（resolution information）

通过服务管理工具实现自动路由事态可以有效减少停机时间，然而，不正确的路由具有相反的效果。异常中的元数据有助于减少不正确的路由。分类对于监控设备判断是否将事态作为

事件发送给服务管理工具十分重要。

解决信息对预防浪费是最有效的。开发人员应该站在运维人员的角度思考："如果以后发生这种情况，我的同事应该怎么去恢复服务？"如果软件这样设计了，就不必在运行阶段再考虑这个问题了。

21.4.3 如何确定异常处理是完整的

对于运维，至关重要的是，所有关键的事态都是基于软件中发生的异常来监控的。然而，事态目录可能并不完整。在最好的情况下，开发人员使用异常文件或数据库来填充事态目录，但是，异常文件里只包含了开发人员在程序里处理的那些错误。确保完整的更好的方法是，在进行主题、史诗或特性编程前进行风险分析，以确定是否捕获了所有可能的错误。

风险分析可以采用石川模型（见图 21-1）。在此分析过程中，DevOps 团队分析可能导致错误路径的原因，然后在主分类中进行搜索（人、机器、度量、方法和材料），其中定义了子分类，最终找到错误的原因。在本例中是因为应用程序需要的一组文件集不完整。

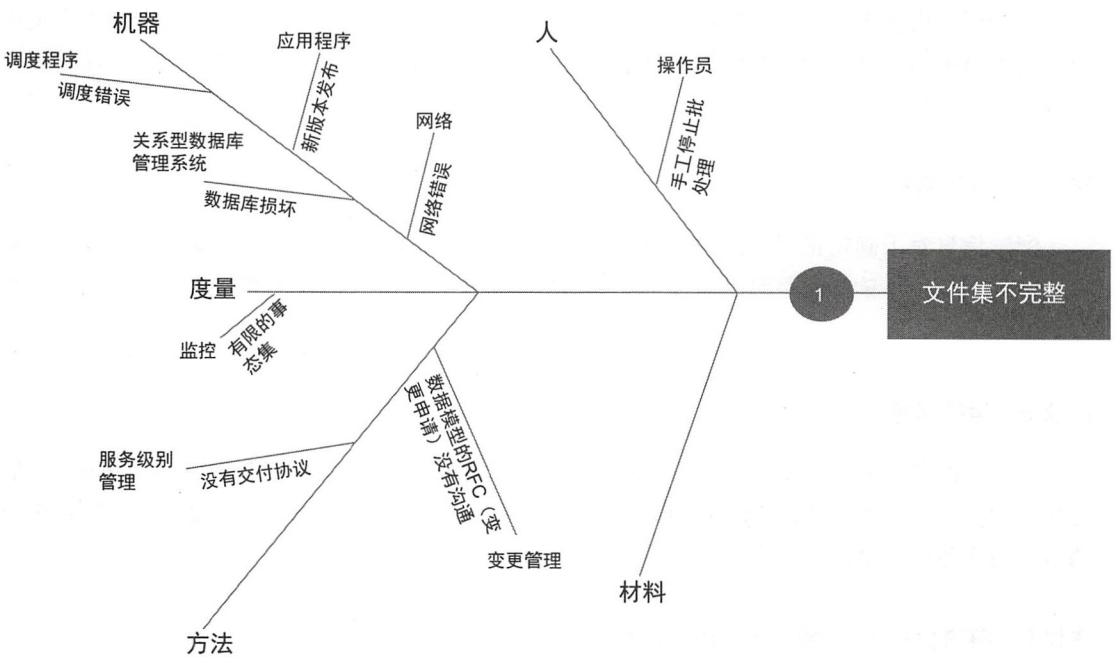


图 21-1 石川模型

接下来为上面的场景填写健康模型。模型采用电子表格的形式，具有如表 21-1 所示的结构。

表 21-1 “文件集不完整”的健康模型

组件	事态	异常 ID	异常	轨迹	监控功能	监控工具	解决方案
预设阶段	1	10001	没有交付协议	Crontab 日志	组件监控	SPS Gensys	发送文件
数据仓库	1	10002	数据库损坏	SQL 日志错误	组件监控	SPS	恢复数据库
Engine B	1	10003	调度错误	调度错误日志	组件监控	SPS Gensys	重新调度任务

基于健康模型，开发人员必须确保开发中考虑了异常处理。应用程序的测试用例至少需要处理以上 3 种错误路径。

21.4.4 在 DevOps 流程中的异常管理

在表 21-2 中定义了异常流程与 DevOps 流程的关系。

表 21-2 在 DevOps 流程中的异常管理

规划	为所有的主题和史诗 • 确定错误路径 • 利用石川图找到导致错误路径的根本原因 • 完成健康模型
编码	为每个特性 • 识别额外的错误路径，识别根本原因（石川分析）和完成健康模型 • 完成事态分类 • 构建单元测试用例来检测将要写入的事态 • 创建包含异常处理的源代码 • 在源代码中添加需要补充的事态并调整石川分析、健康模型和异常文件 • 编写脚本来改变监控的预设
构建	• 编译源代码到目标代码，同时为错误路径运行单元测试用例 • 确保所有的事态能够被单元测试覆盖
测试	• 为错误路径运行涵盖监控预设的系统测试和系统集成测试 • 确保所有事态能够被系统测试和系统集成测试命中
发布	• 确定健康模型、事态分类和异常文件的覆盖率

续表

部署	• 部署目标码、更新监控规则的脚本来更改监控的预设
运维	• 当提供的解决方案无效时，向开发人员反馈
监控	• 当事态没有被正确检测到时，调整监控的预设 • 当没有预见到的事态发生时，请开发人员调整石川分析、健康模型和异常文件

21.4.5 结论

通过异常管理能够大大减少解决事件所需要的时间。DevOps 的工作方法能够更容易地达到该目的。

22

持续集成（#21）

22.1 引言

DevOps 的一个重要的概念叫作软件的持续集成（CI）。本文论述了持续集成的用处和必要性。

22.2 术语

基线

给一组文件设置相同的标记（如用于发布）。

分支

创建分支是一种技术，可对源代码创建两个独立的分支。开发人员在各自负责的分支修改后，会将这些分支再次合并。分支有各种不同的形式，因此，DevOps 团队必须选择合适的分支形式，这种选择也称为分支策略。

签入

提交（commit）、发布签出的源代码。

签出

创建一份工作副本。

存储库

文件和历史文件的集中存储地。

标记

在存储库中对一组相关条目标注版本和发布的标识。

主干

来自存储库的一组文件（也称基线或主线）。

22.3 概念

持续集成

持续集成是一种最佳实践，它允许多个开发人员同时运行同一个应用程序，实现高频地合并源代码，同时对代码进行验证（执行静态测试用例）、编译和测试（执行动态测试用例）。

持续集成这个概念是由 Grady Booch 首次提出的，随后被极限编程采用。持续集成使得程序员不断地将各自的源代码集成到共享的主干中，而且在极限编程中建议每天要集成多次，以避免集成出现问题。

22.4 最佳实践

持续集成是一个灵活的概念。“持续”这个词通常有不同的解释。这主要与签入频率有关。

- 持续集成：每次签入的时候都集成。
- 频繁集成：每天集成一次。
- 集成：根据需要集成。

此外，还有另一个与 CI 不一致的现象，那就是分支。只要分支存在的时间小于持续集成的频率，造成的危害就有限。实际上，任何采用分支的开发团队都不忙着做持续集成。其中，发布分支（release branching）是个例外（参见文章#19）。

本文介绍了持续集成的以下几个方面：

关于 EXIN

国际信息科学考试学会（Exam Institute of Information Science，EXIN）是一家面向全球 ICT 从业人员的中立的考试研发和认证机构。1984 年 EXIN 由荷兰经济事务部创办，后从荷兰政府部门独立出来成立了 EXIN 基金会。EXIN 是全球 IT 服务管理最佳实践知识体系 ITIL®的创始机构之一，开发并拥有 ITIL®版本一和版本二全系列试题的知识产权。EXIN 是全球 ICT 领域内的权威认证机构，个人资质认证体系目前已覆盖 DevOps、数字化服务管理 VeriSM、服务集成管理 SIAM、敏捷开发、精益 IT、IT 服务管理、数据中心管理、项目管理、信息安全、云计算、大数据、IT 资产管理、商业分析、软件测试、数字营销、绿色 IT、OpenStack、安全编程、应用管理等类别。目前，来自全球 165 多个国家和地区的累计近 300 万名 ICT 职业人士已经获得了 EXIN 颁发的资格认证，考试语言多达 25 种。

近年来，EXIN 通过同欧盟政府的紧密合作，在欧洲 IT 人员能力评估框架 e-CF（European e-Competence Framework）中做出了巨大的贡献，并且面向全球的 IT 组织和 IT 专业人士提供基于 e-CF 方面的岗位调研和能力评估测评报告。

联系 EXIN

info.china@exin.com | www.exin.com | 微信公众号: ExamInstitute

EXIN

样题答案表

以下表格为本套样题的正确答案选项，供参考使用。

题号	答案	题号	答案
1	D	14	C
2	A	15	D
3	A	16	A
4	A	17	B
5	B	18	C
6	C	19	D
7	A	20	D
8	B	21	B
9	D	22	C
10	B	23	C
11	D	24	B
12	D	25	D
13	A		

24 / 25

A）错误。维持当前流程不是最好的办法，因为经过一定的发展后，当前流程不再适用。不过，营造一个安全的尝试和实验环境是个比较好的主意。

B）正确。这些活动在成长型企业中效果显著。重新检查流程是有必要的，因为旧的流程不再奏效。通过营造安全的环境进行尝试和实验对于实现持续改进很有必要。（参考文献：*Effective DevOps* ，第二部分和第三部分）

C）错误。重新检查这些流程是有必要的。但是，尽可能减少实验的数量并不是个好主意，不为实验创造条件会阻碍本应发生的持续改进。

25 / 25

A）错误。应当鼓励团队携手合作而非孤身作战。他们彼此需要，也应当相互学习。当人们互相视而不见时，就不可能产生协作。赢得董事会的支持实际上无助于团队本身相互合作。

B）错误。尽管董事会的支持可能使各团队备感受到重视，但这无助于他们相互之间的协作。培训会起到一定的作用，但为了使团队之间更好地合作，他们需要的只是在一起工作。

C）错误。拜访其他公司可能会带来启发，但 DevOps 在不同的企业环境中具有很强的独特性。这对促进协作和减少冲突不会有直接作用。相反，团队之间应当一起工作，共享知识。增加资金可能在极少量人员从事极大量工作时起到一定作用，但在减少冲突和增强协作方面还是远远不够的。

D）正确。这些是最恰当的实践方法，在当前这种情况下，能最大限度地减少运维团队和开发团队之间的冲突，加强双方的协作。（参考文献：*Effective DevOps*）

20/25

A）错误。第二条不是最佳选择，因为它忽略了仪表盘与监控的关键原则。

B）错误。第一条并不是最佳选择，因为你的基础设施应具备的状态需要通过版本控制配置而非变更控制配置来确定。

C）错误。第二条并不是最佳选择，因为忽略了仪表盘的关键原则。

D）正确。为了采用整体方法处理所有基础设施，应当遵循这两条核心原则。（文献：《持续交付》，第十一章）

21 / 25

A）错误。应当立即告知开发部门，使其能够预见潜在的风险和问题。

B）正确。应当立即告知开发部门，使其能够预见潜在的风险和问题。（文献：《企业级 DevOps 的成功之路》）

C）错误。应当立即告知开发部门，使其能够预见潜在的风险和问题。

D）错误。应当立即告知开发部门，使其能够预见潜在的风险和问题。

22 / 25

A）错误。这有助于促进 DevOps 组织的成熟。

B）错误。这有助于促进 DevOps 组织的成熟。

C）正确。这无助于 DevOps 组织的成熟。是否要对会议进行全程记录并再次审查，并没有严格的要求。有必要记录达成共识的内容，而不是记录整场会议。（文献：《持续交付》，第十五章）

D）错误。这有助于促进 DevOps 组织的成熟。

23 / 25

A）错误。文化价值观能提升员工的感受，使工作更有效率，使节奏更为稳定，在犯错后能从中吸取教训。但就其本身而言，对服务水平管理没有帮助。

B）错误。就像 Netflix 开发的“混乱猴子”那样，故意制造混乱可能有助于你着手制定降低风险的措施和恢复方式，但其本身无助于服务水平管理。此外，该系统中的混乱情况应当通过编码来解决，而非作战室这个解决方案。

C）正确。预防、预测及管理基础设施和应用程序的风险是一个系统化的流程，如果不能解决这些风险，可能导致混乱与事端，进而阻碍服务。（文献：《持续交付》，第十一章；《DevOps 企业的成功》）

D）错误。流程主管并没有一个优先级很高的任务来不断检查服务水平管理。至少这不是每个 DevOps 团队的优先选择。

一律的，但 DevOps 应当凝聚团队的力量，为客户注入价值。（文献：*Effective DevOps*，第一章和第二章）

C）错误。工具与自动化都是 DevOps 的重要组成部分，不应忽略。但是，这两者并不一定是首先要关注的内容。相反，更明智的做法是通过创建价值流图及与团队进行商议，努力使整个生产或服务流程更方便、更简短、更廉价。这将开启不可避免的持续改进流程，而这也是很有必要的。其他所有步骤都应当居于次要。

D）错误。在许多企业中，文化转变是必要的。但是，这并不一定是首先要完成的事项，也没有必要召集相关方商讨这一文化上的转变。文化转变应当从如何为客户或相关方创造最大价值开始。要求各个相关方观察价值流图以了解他们所能注入的价值，这是个不错的想法。

18/25

A）错误。让制造问题的人去找出问题可能是最简便的方法，但这没有必要。DevOp 环境中不存在追责。若你的同事不愿承担责任，不要强迫他。强迫任何人做任何事都是失敬的。

B）错误。DevOps 环境不存在追责。若你的同事不愿承担责任，不要强迫他。强迫任何人做任何事都是失敬的。

C）正确。强迫任何人做任何事都是失敬的。犯错是可以接受的。团队成员应共同协作以改正各种错误或应对挑战。（文献：《持续交付》，第三章；*Effective DevOps*，第一部分）

D）错误。这一构建无须修复，你可以回归先前的版本。此外，修复该构建可能不失为一个好的想法，但对犯错的员工施以惩罚却不是个好主意。DevOps 环境中不存在追责。若你的同事不愿承担责任，不要强迫他。强迫任何人做任何事都是失敬的。

19/25

A）错误。INVEST 是创建可维护验收测试套件时推荐的原则。没有具体的信息说明为什么不推荐 ATAM、DIVEST 和敏捷。 敏捷不提供关于测试自动化的任何具体指导或原则。

B）错误。INVEST 是创建可维护验收测试套件时推荐的原则。没有具体的信息说明为什么不推荐 ATAM，DIVEST 和敏捷。

C）错误。INVEST 是创建可维护验收测试套件时推荐的原则。DIVEST 是一个首字母缩略词，在这里被用作干扰项（对应 INVEST）。没有具体的信息说明为什么不推荐 ATAM，DIVEST 和敏捷。

D）正确。验收测试源自验收标准，因此你的应用程序的验收标准的制定应当考虑自动化因素，遵循 INVEST 原则。该原则代表独立、可协商、有价值、可估计、小的与可测试。（文献：《持续交付》，第八章如何创建可维持的验收测试套件）

部署流水线永远比自动化更重要。（文献：《持续交付》，第五章）

D）错误。自动化测试是关键活动。但是，在创建稳定的部署流水线和任务测试自动化之间选择时，你应当始终首先把注意力放在创建稳定的部署流水线上。一旦部署流水线创建完成，就有机会通过测试自动化提升各方面的效率。

15/25

A）错误。维持该流程所需的努力，加入这一复杂事物后可能造成的错误，都使得这一解决方案不是一个好的选择。

B）错误。不同的脚本可能对修改造成困难，从而导致流程内出现问题，不易追踪和解决。

C）错误。在进行构建与部署时不应进行手动调整，因为该流程应当是自动化的，这既是为了提升速度，也是为了避免出错。

D）正确。脚本应当保持一致，才能保证构建和交付流程得到有效测试。环境之间（如 URI 和 IP 等）的差异应当作为配置管理流程的一部分予以处理。（文献：《持续交付》，第六章构建与部署脚本的原则与方法）

16/25

A）正确。金丝雀发布包括向生产服务商中的一小部分推出新版本的应用程序，以快速收集反馈。这样能够快速发现新版本中出现的所有问题，而不会对大多数用户产生影响，因为工作量是逐渐增长的；同时这一做法还能确定响应时间及其他工作表现的衡量标准，减少新版本发布的风险，并且帮助尽快发现与修复漏洞。（文献：《持续交付》，第十章金丝雀发布）

B）错误。在这一情况下，容量测试可能已经自动化，但这些测试的自动化无助于发现这一环境下的故障。

C）错误。这与 DevOps 实践相悖。

D）错误。蓝绿部署需要大量资源，因而在该情境中代价高昂。此外，若需要回滚，在大型数据库中采用这一策略可能导致故障或只读情况发生。另外，这也无助于容量测试效率的提高。

17/25

A）错误。尽管 DevOps 团队在工作过程中沟通很重要，但团队成员此前一直是相互合作的。可以考虑从较易实现的目标入手。更重要的是，必须建立以客户价值为导向的思维方式。一旦形成了这样的思维方式，Em 就能够转而关注对沟通技能的微调。

B）正确。必须简化流程并创建价值流图，以便该团队能够尽可能以最小的努力创造最大的价值。在此之后，应该确定可以改变、应当改变的项目，以及明确已经运作得很好的项目，包括工具、沟通和文化。DevOps 的实践在每家公司的内部体验和表现形式上并非都是千篇

B）错误。（1）并不重要。

C）错误。（4）也很重要。

D）正确。这 3 种做法都是最适用于当前场景的。没有证据表明它们能够在失败的构建中被检查，因此并不具有同样的适用性。（文献：《持续交付》，第三章）

12/25

A）错误。更快速地对测试环境进行部署是可以接受的，这也是持续集成的结果，但这并不会带来任何商业利益。

B）错误。持续集成能帮助他们更快速地向生产环境进行交付，以更小的成本更快地找出缺陷所在，是否使用 Scrum 并不重要。

C）错误。维持发布速度不是 DevOps 的预期效果，更不是持续集成的预期效果。对生产部门增加发布可降低成本，因为此举能够更快速地发现与修复故障。

D）正确。更快速地向生产环境发布是持续集成的一大主要益处，此外还包括更快速地发现故障以减少开发成本和故障修复成本。（文献：《持续交付》，第三章持续集成）

13/25

A）正确。自动化验收测试阶段表明，系统在功能性与非功能性层面上工作正常，在行为上它能满足用户的需求和客户的规格要求。（文献：《持续交付》，第八章）

B）错误。构建测试与单元测试保证新代码本身的合理性。测试并不检查新代码与现有构建的集成程度。

C）错误。这可能是正确的解决方式。但是，在运转的部署流水线中，我们期望验收测试得以自动化。

D）错误。版本控制用于修复失败的构建或解决问题与争议。该项目并不用于表明系统是在功能性还是非功能性层面上工作。

14/25

A）错误。无论何时，所有部署流水线首先应当是单件流的部署流水线，无须自动化就可高效运行。一旦该流水线稳定确立，就有机会选择可行的流程实施自动化。但是，构建稳定的部署流水线永远比自动化更重要。

B）错误。无论何时，所有部署流水线首先应当是单件流的部署流水线，无须自动化就可高效运行。一旦该流水线稳定确立，就有机会选择可行的流程实施自动化。但是，构建稳定的部署流水线永远比自动化更重要。

C）错误。无论何时，所有部署流水线首先应当是单件流的部署流水线，无须自动化就可高效运行。一旦该流水线稳定确立，就有机会选择可行的流程实施自动化。但是，构建稳定的

D）错误。制订计划并确定工作环境要素是不错的想法。不过，除了这种解决方式，还有其他更好的方案，因为暂时停止工作只能在短期内奏效。

8/25

A）错误。这将使受到不良行为影响的员工觉得公司的价值观不太理想、工作环境恶劣，继而引发辞职、工作表现不佳等情况，并且使其他不良行为顺理成章。

B）正确。此人应该很清楚，这样的行为与整个团队和公司政策是相左的。担起责任，帮助他理解为什么要改变及怎么改变，并且让他清楚地知道即便他是首席工程师，若他不改变行为，你势必要让他离开。（文献：*Effective DevOps*，“员工是聪明的也是愚蠢的”一章）

C）错误。这只是在试图掩饰不良行为，同时向工程师释放了一个不良信号，工程师会认为这一行为将被容忍。

D）错误。这可能会暴露某些认为自己没有得到公平对待的员工，也可能被解读为你或公司在试图掩盖什么。

9/25

A）错误。领域一处于第二级水平，领域二处于零级水平。

B）错误。领域一处于第二级水平，领域二处于零级水平。

C）错误。环境和部署已经处于第二级水平，无需更多工作。

D）正确。领域二处于零级水平，首先应当发展到第一级水平，以使该组织能够向第二级水平发展。（文献：《持续交付》，第十五章管理持续交付——配置与发布管理的成熟度模型）

10/25

A）错误。流程主管的职责之一是确保“完成”得以定义。当编码提交而未完成时，应当立即停止工作。但是，编码提交却并未完成，并不一定意味着流程主管表现不佳。

B）正确。工作未能达到“完成”时，意味着还没有足够的顾客价值启动部署流水线。就单件流而言，这将推迟更有工作价值的流。（文献：《持续集成》，第三章）

C）错误。对“完成”下定义是一个项目中需要首先商定的事项之一，而不是在客户会议中才确定。开始编码时，我们应当已经清楚“完成”的定义，否则如何能够知道何时停止编码呢？

D）错误。若编码出现问题或没有注入价值，这便足以成为中断部署流水线并予以修复的理由，或者应当在单件流的流水线中引入更具有价值的工作。

11/25

A）错误。（1）在此处不适用。（3）和（4）也很重要。

献：*Effective DevOps*）

B）错误。有这样一种可能：提高团队多样性可能导致压力和摩擦的增长，因为不同的文化价值观需要相互合作。

C）错误。更大的多样性意味着更多不同的观点，通常这会带来更多创意。

D）错误。这常被看作不利条件（尽管缓慢的决策流程可能是有益的）。更大的多样性意味着可能需要更多时间才能达成共识。

5 / 25

A）错误。这是不够的，因为计划外的非工作时间是工作与生活之间平衡的最大障碍，因此应当尽可能减少。应急计划应当考虑缓解措施，以维持恰当的工作与生活的平衡。

B）正确。实现工作与生活的平衡将使各种类型的人都融入团队中，促进团队的多样性。（文献：*Effective DevOps*，第七章工作与生活的平衡）

C）错误。非工作时间与非标准工作时间可能正适合某些人。但这样的模式对另一些人而言也许颇具挑战性，比如有孩子的员工，但也并非不可能或完全不可调和。“可持续”也可以指提前规划好轮班工作，以使人们能够明确他们的个人需求。如果从事护理行业的人能够“可持续”地推进这一模式，身在 IT 行业的我们也能够做到。

D）错误。非工作时间和非标准轮班工作是任何 7×24 全天候服务模式所采取的工作制度，因此无论是否采用 DevOps，这种工作制度都是现实情况。

6 / 25

A）错误。很多群体都可以非常好地遵循规则，但他们未必被称作团队。

B）错误。很多群体可以举行十分高效的会议，这并不一定是一个团队的标志。

C）正确。一个真正的团队能够维持稳定的工作节奏，并且能够始终向着共同的目标努力。（文献：*Effective DevOps*，第九章）

D）错误。各个团队共同解决问题，并且从不质询团队成员开始。DevOps 拥有一种免责文化。

7 / 25

A）正确。当所有选择都着眼于长远目标时，这是唯一的答案。其他答案都至少有一项短期选择。（文献：*Effective DevOps*）

B）错误。从长远来看，开除领导团队无法解决任何潜在问题，这是权宜之计。逐渐获得平衡才是明智之举，但若没有实施方案，这样的平衡无法自然形成。

C）错误。暂停工作可能是个不错的想法，但仅在短期内有效。心理辅导也很不错，但同样地，不会产生根本的变化，仅是权宜之计。招聘合同制开发人员会有一定作用，是个不错的想法。

答案解析

1 / 25

A）错误。虽然精益、敏捷和 DevOps 是相互关联的，Kaizen 和 5S 并不是启动 DevOps 时的最佳选择。当引入 DevOps 之后，Kaizen 可用于持续改进，5S 通常通过日常良好的实践来维持 Kaizen 改善的效果。但是，这两者都出现在成功引入 DevOps 之后。

B）错误。反馈总是受欢迎的。但这并不能保证在 DevOps 实施时，精益思想能获得最有效的利用。

C）错误。虽然可视化对管理有帮助，但在 DevOps 实施中，它不是最有效的精益实践。

D）正确。创建一个可行的、单件部署流水线将有助于成功实施 DevOps。DevOps 中最重要的工作在于构建从开发部门到运维部门的上游流程，尤其针对单一部署流水线。质量检查是能够达成这一目标的最有效的工作行为。（文献：《企业 DevOps 的成功之路》）

2 / 25

A）正确。ITIL®似乎过于笨重，不适用于 DevOps 的快速处理。轻量型 IT 服务管理是为适应 DevOps 而重组的 IT 服务管理，其重点在于业务连续性，包含一系列最低限度的必要信息。（文献：《企业 DevOps 的成功之路》第 4.3 节 IT 服务管理）

B）错误。目前尚未提出这样的 ITIL®版本。

C）错误。轻量型 IT 服务管理并不是指 ITIL®实施不力，而是“瘦身版 ITIL®”，其焦点在于业务连续性和管理工作量的减少。

D）错误。IT 服务管理针对服务管理而非发布管理。在 IT 服务管理概念中，发布是一种支撑服务的流程。

3 / 25

A）正确。这些原则和系统都是支持 DevOps 实施的相关背景。（文献：《企业 DevOps 的成功之路》

B）错误。在 DevOps 中，云技术能发挥一定的作用，但并非必不可少。

C）错误。持续集成可能是有益的实践，但并非 DevOps 本身知识体系的一部分。

D）错误。项目管理知识体系不是 DevOps 知识体系的一部分。

4 / 25

A）正确。多样性包含各种各样的背景，涉及种族、性别、性取向、阶级、教育水平、语言和工作经历等方面。所有这些独特性都能使一个组织具有更多维的经历与观点可供讨论。（文

- 增加经费以便更好地帮助运维团队与开发团队应对日益增长的需求。

D）

- 建议开发团队和运维团队之间展开实地考察，以建立融洽关系，促进互信与谅解。
- 在开发团队和运维团队之间传播知识，以使他们更有效地合作。

24 / 25

ACMECONST 通过在世界范围内增加雇员和工程队数量积极参与国际市场竞争。该公司也一直在快速扩大客户量，每年增长速度为 30%。

过去一个房间就可以容纳整个工程队的时候，决策相对容易；而现在需要投入大量的时间来做决策，导致整个组织的工作受阻。现在需要多层管理者审批，流程变得更加复杂，导致许多工程师对于整个决策流程大失所望。

目前出现的众多问题的归属也愈发令人困惑，导致决策者常常踌躇不决。工程师还感觉到，额外的流程和官僚主义扼杀了他们的创造力并开始影响他们的士气。

应对这一局面的最好方法是什么？

A）保留当前的流程，但同时明确每个流程的角色、责任和归属，明确权衡生产率与风险的有效方法，促进渐进式变化，并且营造一个安全的环境尝试和实验。

B）重新检查流程，明确哪些事项能够简化，确立每个流程的角色、责任与归属，明确权衡生产率与风险的有效方法，促进渐进式变化，并且营造一个安全的环境尝试和实验。

C）重新检查各个流程，明确哪些事项可以简化，确立每个流程的作用、责任与归属，明确权衡生产率与风险的有效方法，促进渐进式变化，尽可能减少实验的次数，以避免出现不必要的应用程序故障。

25 / 25

在 X-AppGo 公司里，哥伦比亚的运维团队和爱尔兰的开发团队之间存在冲突，这是由于他们有不同的优先级和目标。由于这一冲突的存在，该公司就需要更多的时间和精力解决影响其业务的问题。

为了减少冲突并促进开发与运维团队之间的协作，X-AppGo 应当考虑采取哪些主要措施？

A）

- 如果开发与运维团队愿意，允许他们分别单独开展工作以避免相互冲突。
- 获取董事会对开发与运维团队的全力支持。

B）

- 从公司董事会邀请一名支持者与 DevOps 团队商讨团结协作的重要性。
- 为开发团队与运维团队组织 DevOps 实践培训，以便他们学会通力协作。

C）

- 支持开发团队与运维团队拜访那些 DevOps 做得好的企业。

- 你应当通过当前事件与事件管理，始终了解基础设施的确切状态。

D）

- 你的基础设施应具备的状态需要通过版本控制配置来确定。
- 你应当通过仪表盘与监控始终了解基础设施的确切状态。

21 / 25

当有运维侧变更时，运维部门告知开发部门的最佳时间是何时？

A）无须告知开发部门。运维侧的变更仅运维团队知晓即可。

B）立刻执行。应当尽快通知开发部门。

C）在次日早晨的 Scrum 会议中。

D）当运维团队已经完成验收测试时。

22 / 25

你希望你的 DevOps 组织更趋成熟，有很多方法能做到这一点。

下列哪种方法不会使你的 DevOps 组织更趋成熟？

A）明确定义目标和里程碑，帮助团队成员判断其日常活动是否有价值。

B）明确定义流程，支持并促使团队成员逐日改进流程。

C）对会议的所有内容进行记录，使你的团队成员可以很方便地了解每次沟通的内容。

D）监控并记录每天的活动，以找出小范围内每天取得的进步并予以赞扬。

23 / 25

你为 IT 服务提供商效力。作为你“业务连续性计划”的一部分，你希望确保自己总能达到最低要求的服务水平。

你希望确保 IT 服务的连续性。在 IT 服务连续性管理方面，DevOps 能为你提供哪些帮助？

A）DevOps 的文化价值观“亲和力”与“协作”保证“服务”得到 DevOps 团队成员的高度重视。

B）通过在体系中有意地制造一些混乱，DevOps 实践帮助团队进行日常灾难演练和作战室（Obeya）实践。

C）因为运维部门与开发部门要协同工作，可能需要将降低风险的措施和应急预案进行编码。

D）服务水平管理在 DevOps 中变得更为重要，因为流程主管的任务是监控这一项目。

这样做合适吗？

A）是的。只有破坏构建的人才能够修复它，因此你应当找到负责人，即使这样可能会让人感觉不舒服。

B）是的。你应当始终为破坏构建负责。如果你不负责，你的同事将可能强制执行这项规定。

C）不，DevOps 环境中不存在追责。若同事不承担责任，不要强迫他们。

D）不，你应当首先修复构建，然后抽出时间确定相关责任人并进行处罚。

19 / 25

X-AppGo 开发团队当前在测试中遇到诸多挑战。目前他们使用人工验收测试流程。开发者认为他们所创建的单元测试是十分周密的，可以避免回滚。

在每次发布时，开发团队都需要花费 100 万美元在人工验收测试环节。高级领导层要求开发团队实施自动化验收测试，以降低测试的总成本并尽可能减少引入生产环境中的代码缺陷数量和回滚次数。

在依照自动化需求确定应用程序的验收标准时，应当遵循什么原则？

A）敏捷原则。

B）架构权衡分析法（ATAM）原则。

C）DIVEST 原则。

D）INVEST 原则。

20 / 25

你希望采用整体方法管理所有 IT 基础设施。

哪两条原则能在这方面帮助你？

A）

- ❍ 你的基础设施应具备的状态需要通过变更控制配置来确定。
- ❍ 你应当通过监控与事件管理，及时了解基础设施的确切状态。

B）

- ❍ 需要通过变更控制配置来确定你的基础设施应具备的状态。
- ❍ 你应当通过仪表盘及事件管理，始终了解基础设施的确切状态。

C）

- ❍ 你的基础设施应具备的状态需要通过版本控制配置来确定。

17 / 25

S 公司是一家中型汽车零部件供应商，为一家大型企业 T 汽车公司提供产品。该公司向 T 汽车公司供应汽车零部件，这部分销售额占其总销售额的 60%。

董事会召开会议，商讨新的合作方式。T 汽车公司要求 S 公司将其交付方式转变为即时交付，否则将中断与 S 公司的业务往来。失去这个客户意味着 S 公司将无以为继，因此该公司必须尽快转变为即时交付。这一转变必须在 6 个月内完成，因此最多只有 5 个月的时间准备。

其中需要实施的一项工作是通过射频识别（RFID）技术追踪各个零部件，这将有助于保持生产流程的透明化。快速回顾当前的流程有助于尽快向 RFID 支持流程转变。

首席信息官应控制转变流程。她认为如果采用 DevOps 方法来实现最低限度发布是可以做到的。原则上应当首先确立 RFID 的生产理念，最后一步是使用 RFID 数据的生产控制系统应当得以运行，但是已经没有足够的时间按先后顺序来完成这些步骤了，因此这 3 项工作必须同时执行。

首席信息官指派 Scrum Master Em 先生来负责这一项目。开发团队准备构建部署流水线。Em 认为开发团队充满热情、工作努力，但应更加自律。此外，发布频率应当提高。

Em 应当首先关注什么？

A）Em 应当把注意力集中在沟通方面，这是 DevOps 实践中最重要的元素。Em 首先应当做的是打破团队中的僵局，并且明确沟通规则。

B）Em 应当首先与团队商议如何编制价值流图，并构建单件流，因为构建流程和精简进程都很重要。

C）Em 应当首先与团队成员讨论基础设施与工作环境问题，因为在所有工具和实践方法都良好运作时，DevOps 效率最高。

D）Em 应当首先召集所有相关方，对他们进行有关 DevOps 的培训，并请他们协助传播企业文化方面的调整，因为文化调整是 DevOps 必不可少的部分。

18 / 25

贵公司正在尝试转变，并且开始使用 DevOps 的方式开展工作。你的团队也在经历这一转变，你正在参与讨论代码提交阶段的最佳实践。

你的同事孙说道：“当某一构建遭到破坏且无人担责时，我们应当找出造成破坏的人并要求他们展开工作，以保证他们能修复这一构建。”

有一名团队成员说道："这条部署流水线最需要的是自动化。我们首先要做的是让它自动化起来。"

这种说法对吗？

A）是的，这是正确的。部署流水线自动化是提升效率最重要的因素。
B）是的，这是正确的。关注部署流水线自动化的创建，能克服之后可能遇到的潜在问题。
C）不，这是错误的。完成单件流及一个可靠的部署流程是优先级最高的任务。该流程的自动化可以暂缓实施。
D）不，这是错误的。首先应当自动化的是测试流程而非部署流水线。

15 / 25

DevOps 的最佳实践是利用同一流程针对应用的不同运行环境进行部署，这将确保构建得到有效测试。你使用脚本去构建和自动化部署流水线。

完成这一任务的最好方法是什么？

A）每种环境采用一个脚本，并将其作为版本控制系统的一部分加以维护。
B）不同环境使用不同的特定脚本，以解决环境之间的差异问题。
C）不同环境采用同样的脚本，对特定的配置采用手动调整参数。
D）采用同一脚本在不同环境中进行部署，并且单独管理配置信息。

16 / 25

AppBC 公司正在采用 DevOps。该公司实施了持续部署，并且具备高度自动化验收测试和每日向生产部交付新软件的稳定部署流水线。

AppBC 公司有一个巨大的数据库及众多用户。该公司有一个全面可靠的容量测试策略。由于该公司环境范围大而复杂，随着每个新版本的发布，生产部就会出现一些小故障。

什么策略能够最有效地帮助 AppBC 预防这些故障？

A）采用金丝雀发布。
B）自动化容量测试。
C）降低交付率。
D）采用蓝绿部署。

这 4 种做法哪一个最适用于分布式地点解决当前的主要痛点？

A）（1）和（2）

B）（1），（2）和（3）

C）（2）和（3）

D）（2），（3）和（4）

12 / 25

一个开发团队对 DevOps 感兴趣，其主要感兴趣的对象是持续集成（CI）。他们目前开发并维护着 3 种主要解决方案及 4 种次要解决方案。他们采用 Scrum 实践方法。每次冲刺都需要 4 周时间，平均每 10~15 天对测试环境都有一次发布，平均每个月对生产环境有一次发布。他们想向管理层提交一个定性的商业论证，来支持他们为创建持续集成实践模式而付出的投资与努力。

持续集成有哪些显性收益对该商业论证最为有利？

A）每天对测试环境进行一次部署能极大地提高商业效益并大大缩减开发成本。

B）这有助于提升团队士气。由于公司已经在使用 Scrum，持续集成将为公司业务带来显著的益处。

C）它通过更好的集成测试提高了业务稳定性，同时维持发布速度以防止产生额外成本。

D）在生产环境中，每天进行一次信息发布能够提升业务收益，并大大减少开发成本。

13 / 25

考虑对基本部署流水线进行具体解析。

哪个阶段表明该系统在功能性与非功能性层面均发挥作用？

A）自动化验收测试。

B）构建与单元测试。

C）手动验收测试。

D）版本控制。

14 / 25

一位首席信息官将她最信赖的员工——担任 Scrum 主管的迈克尔指派给某个项目。开发团队打算构建一条部署流水线。

迈克尔相信开发团队的好意与主动性，但希望他们更自律。此外，发布频率也应有所提高。迈克尔希望开发团队能更加频繁地发布。

依据上述信息评定这两个领域的成熟度，接着你应提出改进建议。

在这两个领域中，公司建设者应选择哪一个来进行持续改进才能提升至第二级水平？

A）环境与部署，构建管理及持续集成都处于零级水平。这项工作应在两个领域中同时进行。
B）环境与部署、构建管理及持续集成处于第一级或第一级以上水平。其他领域也应开展工作以获得提升。
C）环境与部署处于零级水平，构建管理和持续集成处于第一级水平。重点应当首先放在环境和部署上。
D）环境与部署处于第二级水平，构建管理与持续集成处于零级水平。重点只须放在构建管理与持续集成上。

10 / 25

你的团队需要为新产品开发一个部署流水线。作为持续集成的一部分，你需要明确定义流水线的提交阶段。

你与团队成员讨论这一阶段时，流程主管说道："完成"一词的定义应在提交阶段开始之前或进行时确定。若编码在提交时未能达到定义的"完成"，该工作就应停止。

是这样吗？

A）是的。若这项工作未能完成，就意味着流程主管未能履行职责，应当立即予以解决。
B）是的。尚未完成的工作不得提交，因为这样的工作不会给客户带来增值。
C）不是，"完成"的定义只能在客户会议上确定。等待会大幅减缓工作进度。
D）不是，应持续进行部署流水线上的工作。如果编码没有完成，只须使之处于非激活状态。

11 / 25

一个跨国企业要将分布在各地的任务合并提交到总部位于得克萨斯州达拉斯的中央代码库，这面临着诸多挑战。这些地方包括墨西哥城、巴黎、圣迭戈和英格兰地区。这些地区何时提交并无规律可循，有时人们并不清楚测试是否有失败的。

可行的做法有以下 4 种：
（1）失败的构建不提交。
（2）在提交之前务必在本地运行所有测试，或利用持续集成服务器替你完成这一工作。
（3）待提交测试通过后再进行下一步工作。
（4）不要注释失败的测试。

○ 制订解决这些问题的计划。

8 / 25

你一直在一家十分成功的创业公司里领导着一支小型的 IT 团队，该公司最近被一家大型的企业收购了。作为被收购的一部分，你的团队提供更多新型的解决方案，规模也显著扩大，团队成员中既有来自收购企业的员工，也有你精挑细选的新员工。

在此次并购之后，你开始发现团队内的关系变得紧张起来，这一点在来自收购企业的团队成员身上表现得尤为显著。你开始展开调查，发现问题出在你这一方的技术工程师身上。他是你在创业期聘用的第一名员工，也是你最关键的系统中最具有专业知识的员工。他是一名出色的工程师，但他的态度有些问题。他最大的问题是总会说些带有性别歧视与偏见的笑话。他很具有竞争意识，不喜欢按照流程行事。

你知道他总是如此，你的初创团队也认为这很“正常”，于是这样的行为得到了默许。由于新的团队更加多元化，这种行为造成团队成员对其排斥、不适应及其他种种问题。

应对这种局面的最好办法是什么？

A）告知各位同事，他们有两种选择：要么被解雇，要么忽略此人的缺点，因为你没有其他人可替换。

B）告诉工程师他可以做出选择：要么被解雇，要么在其他人的帮助下（如果他需要的话）改变自己的行为方式。

C）和工程师谈话，请他注意不要与新同事开玩笑，但旧同事无妨，因为他们已经习以为常了。

D）与团队对话，要求每位成员宽容他们的同事并集中精力完成工作。

9 / 25

你目前正在为一家数年前采用了 DevOps 实践方法的大中型企业评估公司建设者。

他们聘请你为他们当前的成熟度做一个评判。完成此项工作后，你应提出改进建议。他们希望了解应当重点关注哪一领域才能达到下一个成熟阶段，即第二级——量化管理。

你发现多数领域都达到第一级——一致，但其中有两个例外：

一是环境与部署。该领域负责精心策划部署，并且检测发布与回滚流程。

二是构建管理与持续集成。在该领域，你发现在定期的自动化构建与测试中，任何构建都可以通过源代码管理，采用自动化流程重建。

你觉得有什么确切的特征表明这是一支团队而不只是一个小组呢？

A）该团队遵守在团队会议中共同制定的规则。
B）该团队召开的高效会议都是由自己主持的。
C）该团队以稳定的工作节奏朝着共同的目标推进。
D）该团队通过质询负责某项工作的团队成员的方式来解决问题。

7 / 25

AppAtoZ 公司正以惊人的速度发展壮大，致力于为苹果和安卓平台开发和部署移动应用程序。

为使其当前的移动应用程序尽可能快速地提升功能，这家创业企业的开发团队成员经受了巨大的压力。在过去的 6 个月里，他们平均每周工作 60 小时。领导层不愿雇用更多员工，他们更关心的是如何在提高收益的同时降低运维与开发成本。最近几个月，旷工、打电话请病假的员工均有所增加，有的员工甚至辞去了 AppAtoZ 的工作，从而导致现有员工的工作量剧增。雇用新员工、缩短新员工的上手时间并不能快速缓解开发团队的工作压力。

要解决员工倦怠与压力的问题，应当考虑哪些长远的战略？

A）
- ○ 应在团队中同时采用永久雇佣和合同工制，以更好地控制工作量。
- ○ 管理层与开发部门应当找出工作环境中导致员工倦怠的因素。
- ○ 制订解决这些问题的计划。

B）
- ○ 解雇领导团队，原因是他们无法胜任工作。
- ○ 聘用一支更具有能力的领导团队，他们知道如何促使工作与生活达到平衡，并且创造可持续的、切合实际的企业文化。
- ○ 为开发团队提供时机，使其逐渐达到工作与生活之间的平衡。

C）
- ○ 让开发团队成员暂时停止工作。
- ○ 招聘合同制开发人员，以应对每年高峰时期增加的开发工作量。
- ○ 鼓励开发人员在必要时寻求专业的心理援助。

D）
- ○ 让开发团队成员暂时停止工作。
- ○ 让领导层与开发部门找出工作环境中可能导致倦怠的所有因素。

B）

- ○ 敏捷
- ○ 持续集成
- ○ 持续交付
- ○ 云技术

C）

- ○ 持续集成
- ○ 持续交付
- ○ ITIL®

D）

- ○ 项目管理知识体系（PMBoK）
- ○ 架构，持续交付
- ○ ITIL®

4 / 25

通过吸纳具有更广泛的个人背景和文化的人才来提高团队的多样性，这种做法有哪些显著的益处？

A）能够带来更多的经验与观点。

B）使得团队内部的摩擦减少。

C）限制了创造力的发挥，也限制了人们提出新见解的能力。

D）使得达到某一决策点需要更长时间。

5 / 25

DevOps 关注可持续工作实践的实现，而工作与生活的平衡是其关键组成部分。

工作与生活平衡的真正意义是什么？

A）为计划外的加班提供报酬便足以形成工作与生活的平衡。

B）强调通过恰当的计划达到工作与生活的平衡，有助于培养出一支多样化的团队。

C）无论对谁而言，加班和非标准的轮班与工作和生活的平衡都是格格不入的。

D）在一支多样化的团队中应当避免加班和轮班，以维持工作与生活的平衡。

6 / 25

你认为自己的开发团队是一支真正的团队。

样题

1 / 25

首席技术官认为，在实施 DevOps 时，最有效的方法是将精益的相关概念应用在 DevOps 过程中。

请问，在引入 DevOps 时，精益管理的哪些原则或实践方法最有效？

A）Kaizen（专有名词，意为小的、不花钱的持续改善）与 5S。由于敏捷和 DevOps 是以精益的核心思想为基础的，而持续改善与 5S 是精益的基础，因此在引入 DevOps 时，使用这些方法最为有效。

B）先实施 Kaizen。DevOps 要求运维团队对开发团队进行持续反馈。先实施 Kaizen 能够形成上溯反馈环路，有助于这一原则在 DevOps 实施中应用。

C）Obeya 系统。DevOps 整合了不同的管理流程。Obeya 系统提供了整个流程的可视化，为成功引入 DevOps 创造了条件。

D）单件流与质量检查。DevOps 从上溯流程和单一价值流的构建中获益。单件流实现了这一点，而质量检查有助于改善/简化和实施该流程。

2 / 25

什么是轻量型 IT 服务管理？

A）以业务连续性为中心的 IT 服务管理。

B）新一版标准化的 ITIL®。

C）ITIL®流程实施不力。

D）以发布管理为导向的 IT 服务管理。

3 / 25

哪种知识体系最能体现 DevOps 原则？

A）

- 敏捷
- 持续交付
- IT 服务管理
- 精益管理/丰田生产体系

第三部分 EXIN DevOps Master 认证样题 & 解析

说明

本试卷是基于 EXIN DevOps Master 的，EXIN 考试准则适用于本考试。

本试卷由 25 道单项选择题组成。每道选择题有多个选项，但这些选项中只有一个是正确答案。

本试卷的总分是 25 分，每道题的分数是 1 分。每答对 1 题获得 1 分。如果你获得的总分数为 17 分或以上，你就通过了本考试。

考试时间为 60 分钟。

祝你好运!

续表

考试要求	考试内容	文献	文献参考
4	4.1	B	第 11，12，13 章
	4.2	B	第 2，14 章
	4.3	A	第 4，5，14，16 章
		B	第 11 章
	4.4	C	第 2，4 章
	4.5	A	第 14，15，16，17 章
5	5.1	C	第 7 章

注：阅读《凤凰项目》对以下内容的理解将特别有帮助：

1.1，1.2，3.1，3.3，3.4，4.4。

文献考点分布矩阵

考试要求	考试内容	文献	文献参考
1	1.1	A	第 1，2，3 章
		B	第 1 章
		C	第 1，2，3 章
	1.2	A	第 6，7，8，9，10，11，12 章
		C	第 5，6 章
	1.3	A	第 4，5 章
		C	第 4 章
2	2.1	C	第 7 章
	2.2	C	第 5，7 章
	2.3	A	第 3，4 章
		B	第 11 章
		C	第 5，7 章
	2.4	C	第 7 章
	2.5	B	第 4 章
3	3.1	A	第 16 章
		B	第 3，15 章
		C	第 4 章
	3.2	B	第 5，6 章
		C	第 5 章
	3.3	B	第 10 章
		C	第 8 章
	3.4	C	第 4，7 章
	3.5	A	第 4，11，12，13 章
		B	第 3，4，5，6，7，8，9 章

原出版社：IT Revolution Press (2013 年 1 月 10 日)

中文版书号：ISBN 978-7-115-40365-0

强烈建议在培训之前阅读本书。

E 《**DerOps 实践指南**》（***The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations***）

作者：Gene Kim, Jez Humble, Patrick Debois, John Willis

英文版书号：ISBN-13: 978-1942788003；ISBN-10: 1942788002

原出版社：IT Revolution Press (2016 年)

中文版书号：ISBN 978-7-115-48017-0

F 其他资源

http://newrelic.com/devops

http://devops.com/

4 文献

必选教材

A ***Effective DevOps：Building a Culture of Collaboration, Affinity, and Tooling at Scale***

作者：Jennifer Davis, Katherine Daniels

英文版书号： ISBN-13：978-1491926307；ISBN-10:1491926309

出版社：O'Reilly Media (2016 年 6 月 25 日)

B 《持续交付：发布可靠软件的系统方法》（***Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation***）

作者：Jez Humble, David Farley

英文版书号：ISBN-13: 978-0321601919；ISBN-10: 0321601912

原出版社：Addison-Wesley Professional（2010 年 8 月 6 日）

中文版书号：ISBN 978-7-115-26459-6

C 《企业 **DevOps** 的成功之路》（***Success with Enterprise DevOps***）

作者：Koichiro (Luke) Toda, Nobuyuki Mitsui

本书中文版电子书可在 EXIN 官方平台免费获取。

可选教材

D 《凤凰项目：一个运维的传奇故事》（***The Phoenix Project***）

作者：Gene Kim, Kevin Behr, George Spafford

英文版书号：ISBN-13: 978-0988262577；ISBN-10: 0988262576

续表

英文	中文
Plan-Do-Check-Act cycle (PDCA cycle)	计划—实施—检查—改进环（PDCA 环）
Post-Mortem	事后剖析
Product Owner (in Agile Scrum)	产品负责人
Retrospective	回顾
Rhythm (in Lean)	节奏
Scaling (of DevOps or Agile Scrum)	规模化
Scrum	Scrum（无中译文）
Scrum Master (in Agile Scrum)	Scrum Master（无中译文）
Service Deployment	服务部署
Service Level Agreement (SLA)	服务级别协议
Sprint	冲刺
Test Story	测试场景
Test-Driven Development	测试驱动开发
Tools	工具
Toyota Production System (TPS)	丰田生产模式
User Story	用户故事
Velocity (in Agile Scrum)	速度
Version Control	版本控制
Waterfall	瀑布式
WiP-limit	在制品数量限制
Work-in-Progress (WiP)	在制品

3 基础术语表

应理解以下基本概念。建议学员对这些概念进行调查研究。

英文	中文
Affinity (in DevOps)	亲和
Agile	敏捷
Application Deployment	应用部署
Artifact Management	构件库管理
Automation	自动化
Blamelessness	免责
Cloud Computing	云计算
Collaboration (in DevOps)	协作
Configuration Management	配置管理
Containers	容器
Continuous Delivery	持续交付
Continuous Deployment	持续部署
Continuous Integration	持续集成
Definition of Done (in Agile Scrum)	完成的定义
Infrastructure Automation	基础架构自动化
Iteration	迭代
ITSM (IT Service Management)	IT 服务管理
Ji-Kotei-Kanketsu (JKK)	质量检查
Just-in-Time (JIT)	准时制
Kaizen (in Lean)	持续改善
Lean	精益
Micro-service	微服务
Minimum Viable Product	最小可用产品
One-piece-flow	单件流
Operations Story	运维场景
Organizational Learning	组织级学习

学员能够

4.3.1 说明在实现有效的 DevOps 的过程中，何时需要向基于云的基础设施转移，以及何时不需要进行此类转移。

4.3.2 在 DevOps 内应该如何对基于云的基础设施进行管理。

4.4 业务连续性 2%

学员能够

4.4.1 利用 DevOps 促进业务连续性实践。

4.5 规模化 4%

学员能够

4.5.1 进行情景分析，说明为什么在这种情况下扩大规模或减小规模是非常重要的，找到最佳的实施方法。

4.5.2 对规模化过程中出现的问题进行情景分析，并且找出解决该问题的最佳方法。

4.5.3 社会政策和雇佣实践如何支持 DevOps 的规模化。

5. 生命周期结束 **2%**

5.1 一个产品或服务的生命周期结束的条件。 2%

学员能够

5.1.1 说明在终止一项服务或一个产品之前应该满足哪些条件。

3.5 自动化、工具和测试 6%

学员能够

3.5.1 说明为什么自动化对有效的 DevOps 是非常重要的。

3.5.2 说明如何使用工具提高 DevOps 的整体效率。

3.5.3 说明如何使用工具支持 DevOps 思维模式和文化。

3.5.4 说明为什么 DevOps 测试自动化非常重要。

3.5.5 进行情景分析，选择正确的方式来实现验收测试自动化。

4. 运维和规模化 **22%**

4.1 管理数据、基础设施和环境、组件和依赖性 10%

学员能够

4.1.1 说明在 DevOps 中管理数据库中的数据时会遇到哪些问题。

4.1.2 对 DevOps 中数据库的使用进行情景分析，并且对遇到的问题提供最佳的解决方案。

4.1.3 为实施一个部署需要准备一个基础设施环境，并且在部署后需要对其进行管理。对此进行情景分析，找出进行此项工作的最佳方法。

4.1.4 进行情景分析，提出一个管理组件的通用策略。

4.1.5 说明如何管理依赖性。

4.2 配置管理和版本控制 4%

学员能够

4.2.1 说明为什么版本控制是有效的 DevOps 的关键。

4.2.2 说明如何对数据、基础设施和组件进行版本控制。

4.2.3 进行情景分析，针对配置管理问题提出一个最佳策略。

4.3 云和不可变的基础设施 2%

3. 开发和部署 **30%**

3.1 持续交付和持续集成 12%

学员能够

3.1.1 说明为什么持续交付对于有效的 DevOps 至关重要。

3.1.2 在情景中分析如何整合持续交付。

3.1.3 在情景中分析如何采用持续交付解决问题。

3.1.4 说明为什么持续整合对于有效的 DevOps 至关重要。

3.1.5 通过对一个分布式团队或分布式控制系统进行情景分析，分析如何实现持续整合。

3.1.6 在情景中分析如何采用持续整合解决问题。

3.2 部署渠道 4%

学员能够

3.2.1 说明一个 DevOps 部署渠道的分解逻辑。

3.2.2 说明如何使用创建和部署脚本。

3.3 持续部署 4%

学员能够

3.3.1 说明为什么要针对有效的 DevOps 对迭代计划和发布计划进行修改。

3.3.2 在情景中分析如何实施持续部署。

3.4 质量检查、节奏、在制品和单件流 4%

学员能够

3.4.1 说明质量检查、节奏、在制品和单件流的概念。

3.4.2 在情景中分析质量检查、节奏、在制品和单件流遇到的具体问题，并且找到适合的解决方案。

2. 计划、要求和设计 **18%**

2.1 应用或服务生命周期管理 4%

学员能够

2.1.1 说明 DevOps 如何为应用生命周期管理带来价值。

2.1.2 说明 DevOps 在用于服务生命周期管理时如何改善客户体验。

2.2 项目章程（定义范围）和视觉控制 4%

学员能够

2.2.1 说明应该如何确定一个 DevOps 项目的范围。

2.2.2 为什么说 DevOps 项目的视觉控制能够促进 DevOps 实践。

2.3 基础设施和架构设计 4%

学员能够

2.3.1 说明 DevOps 如何改变或影响 IT 基础设施和架构的设计。

2.3.2 说明为什么云计算和虚拟化技术能够使 Dev 和 Ops 实现更轻松的集成。

2.4 服务等级要求和协议 2%

学员能够

2.4.1 说明 DevOps 如何改变服务等级要求和协议。

2.5 实施测试策略：用户场景、测试场景和运维场景 4%

学员能够

2.5.1 说明在向 DevOps 转变时为什么需要对测试策略进行修改，以及如何修改。

2.5.2 分析用户场景、测试场景和运维场景。

考试内容

1. DevOps 应用 **28%**

1.1 DevOps 理念与益处 10%

学员能够

1.1.1 在情景分析中对 DevOps 进行反模式分析。

1.1.2 说明 DevOps 的优势。

1.1.3 说明为什么 DevOps 非常适合当前的软件开发流程。

1.1.4 说明为什么 DevOps 需要一个特定的思维模式。

1.1.5 说明 DevOps 如何适合精益和敏捷开发实践。

1.2 组织文化 12%

学员能够

1.2.1 说明为什么 DevOps 的四大组成部分（协作、亲和力、工具和规模化）如此重要。

1.2.2 针对 DevOps 思维方式缺失的部分进行情景分析。

1.2.3 说明如何通过促进协作、DevOps 思维模式、共鸣及信任，在一组人中创建一个团队。

1.2.4 关于协作的错误概念的情况分析，并且找出正确的解决办法。

1.2.5 关于冲突管理的情况分析，并且找出最佳的解决方案。

1.2.6 说明人力资源管理如何能够促进多样化，以及这样会给企业带来哪些好处。

1.3 DevOps 原理和概念 6%

学员能够

1.3.1 说明不同软件开发方法（瀑布式、敏捷、Scrum）及其基本原理的使用和实用性。

1.3.2 说明不同运维方法（IT 服务管理）的使用和实用性。

1.3.3 说明精益系统方法的使用和实用性。

2 考试要求和内容

考试要求		考试内容	比例
1. DevOps 应用			28%
	1.1	DevOps 理念与益处	
	1.2	组织文化	
	1.3	DevOps 原理和概念	
2. 计划、要求和设计			18%
	2.1	应用或服务生命周期管理	
	2.2	项目章程（定义范围）和视觉控制	
	2.3	基础设施和架构设计	
	2.4	服务等级要求和协议	
	2.5	实施测试策略：用户场景、测试场景和运维场景	
3. 开发和部署			30%
	3.1	持续交付和持续集成	
	3.2	部署渠道	
	3.3	持续部署	
	3.4	质量检查、节奏、在制品和单件流	
	3.5	自动化、工具和测试	
4. 运维和规模化			22%
	4.1	管理数据、基础设施和环境、组件和依赖性	
	4.2	配置管理和版本控制	
	4.3	云和不可变的基础设施	
	4.4	业务连续性	
	4.5	规模化	
5. 生命周期结束			2%
	5.1	一个产品或服务的生命周期结束的条件	
总计			100%

培训时长根据完成的实践任务的工作量多少有所不同。培训期可能包括但不限于：

- 为期 2 天的面对面理论培训和为期 1 天的实践培训。学员对实践作业做好准备，并且在培训开始之前复习好相关文献。
- 实践任务的在线培训和评估。学员通过电子学习渠道学习，并且针对实践作业做好准备。培训师评估实践作业的工作量。
- 为期 2 天的面对面理论培训和为期 3 天的实践培训，学员无须做准备。

培训小组人数

建议人数为 16 人（不适用于在线培训课程）。

培训时长

本培训课程要求的培训时长不少于 24 小时。该时长包括学员分组、备考和短暂休息，不包括做家庭作业、备考的准备工作，以及午餐及休息时间。

建议个人学习量

120 小时，根据已掌握的知识的情况可能有所不同。

样题

授权机构可通过 EXIN PartnerNet 平台，下载最新的认证样题。

授权培训机构

可通过 EXIN 官网 www.exin.com 查找本认证的授权培训机构。

- EXIN Application Management Foundation 认证：能够让你更好地理解应用的运行和支持环境。

认证要求

- 接受经授权的 DevOps Master 培训，包括成功完成实践作业（Practical Assignments）（实践作业可通过凤凰项目沙盘演练来完成）。
- 顺利通过 DevOps Master 考试。

考试细节

考试类型：计算机考试或笔试，单选题

题目数量：50 道

通过分数：65%

是否开卷考试：否

是否允许携带电子设备/辅助设备：否

考试时间：120 分钟

EXIN 的考试规则和规定适用于本类考试。

培训

参加授权培训是获得该认证的必要条件。学员应掌握 DevOps 原理，以及精益和敏捷概念的基本知识。这些知识可通过以下渠道获得：

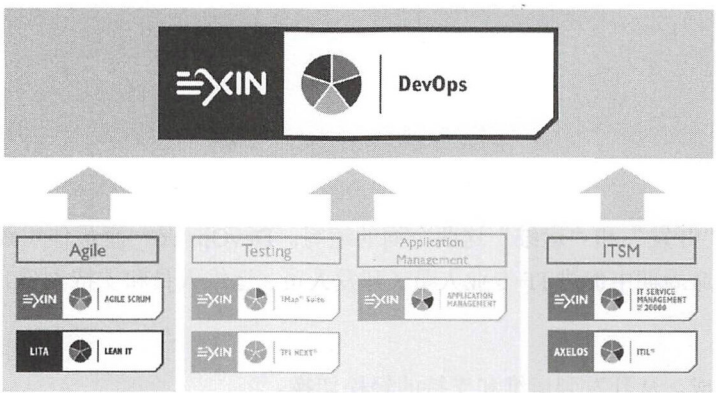
- 电子学习（e-Learning）。
- 额外参加一天的 DevOps 入门培训。

或者

- 阅读《凤凰项目》（*The Phoenix Project*）。

背景信息

EXIN DevOps 认证项目如下所示。



目标群体

DevOps 不仅在软件开发领域享有盛誉，其法则也同样被应用到 IT 服务项目和其他项目中。DevOps Master 培训和认证项目的目标人群是希望掌握 ICT 管理最新发展动态的所有专业人员。

在 DevOps 团队中工作的任何人，或者正考虑转向 DevOps 的企业及该企业内的人都将受益于 EXIN DevOps Master 认证项目。

目标群体包括（不限于）：应用或服务产品经理、敏捷项目经理、敏捷 Scrum Master、项目经理、测试工程师、测试经理、IT 服务经理、IT 流程经理、精益 IT 从业人员。

由于该认证属于高阶认证，要考察申请者是否具备一定的实践经验和行业知识，因此，我们强烈建议你了解和掌握 DevOps 相关应用领域的知识与经验，包括：

- EXIN Agile Scrum Foundation 认证：能够让你更好地理解 DevOps 工作方式的敏捷性。
- TMap Suite® Text Engineer 认证：能够让你更好地理解每个环节中自动化测试和集成测试的环境。
- EXIN IT Service Management Foundation 认证：能够让你更好地理解应用或服务的运行环境和支持环境。
- LITA Lean IT Foundation 认证：能够让你更好地理解一次准确性（first-time-right）（防止生产/运行环境中错误的出现）和其他精益概念。

第二部分 EXIN DevOps Master 认证备考指南

1 概述

综述

DevOps 是“开发”和“运维”这两个词的缩写。DevOps 是一套最佳实践方法论，旨在在应用和服务的生命周期中促进 IT 专业人员（开发人员、运维人员和支持人员）之间的协作和交流，最终实现：

- 持续集成。从开发到运维和支持的轻松切换。
- 持续部署。持续发布，或者尽可能经常发布。
- 持续反馈。在应用和服务生命周期的各个阶段寻求来自利益相关方的反馈。

DevOps 改变了员工的工作思维方式。DevOps 重视所做工作的多样性，支持企业为加快实现业务价值而创建的流程，并且评估社会和技术变革产生的影响。DevOps 是能够让企业和个人建立和保持可持续实践的一种思维方式和工作方式。

成功的 DevOps：

- 能创造一个轻松的没有相互指责的企业文化环境，可以与他人分享自己的经验，并且达成共识，使人员和团队能够以持续有效的方式发挥他们的特长。
- 为企业准时制（Just-in-Time , JIT）生产策略提供应用和服务。
- 通过基于风险的业务需求方案来确保 IT 服务的连续性。
- 管理应用和服务的整个生命周期，包括生命周期的结束条件。

该认证主要在理论知识的基础上增加实践技能，使 DevOps Master 能够成功地应用于一个企业团队中，并且促成其实施法则在企业组织中执行。

该认证的研发集结了 EXIN 全球智库（EXIN Professional Group, EPG）中致力于 DevOps 工作领域的各方面专家，共同协作完成。

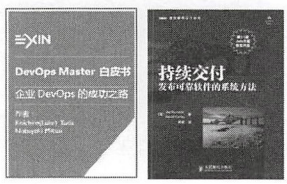
3 更多 DevOps 学习资源

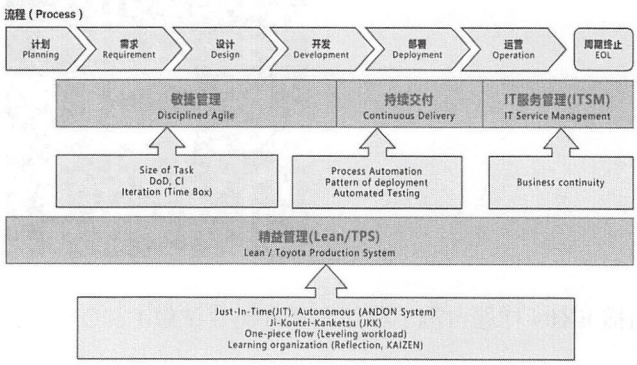
扫描 EXIN 公众号，下载全套 DevOps 认证免费学习资源。

扫描 EXIN 样题自测小程序，在线免费试做 DevOps 模拟题。

2 DevOps 系列认证模块

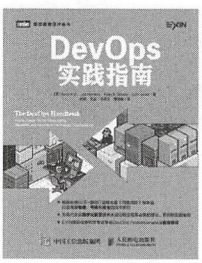
DevOps Master

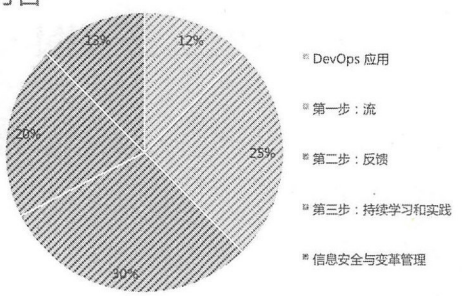


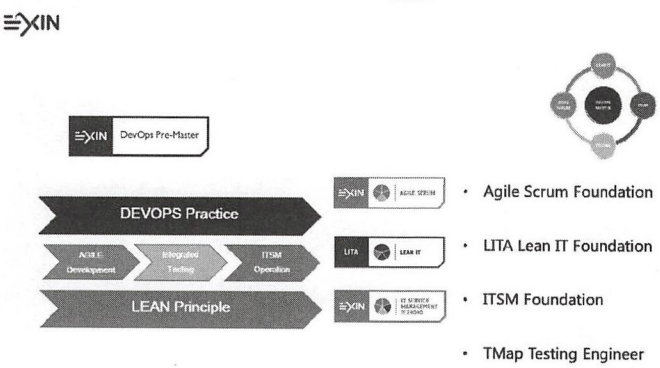


DevOps Professional

核心教材 & 考试内容

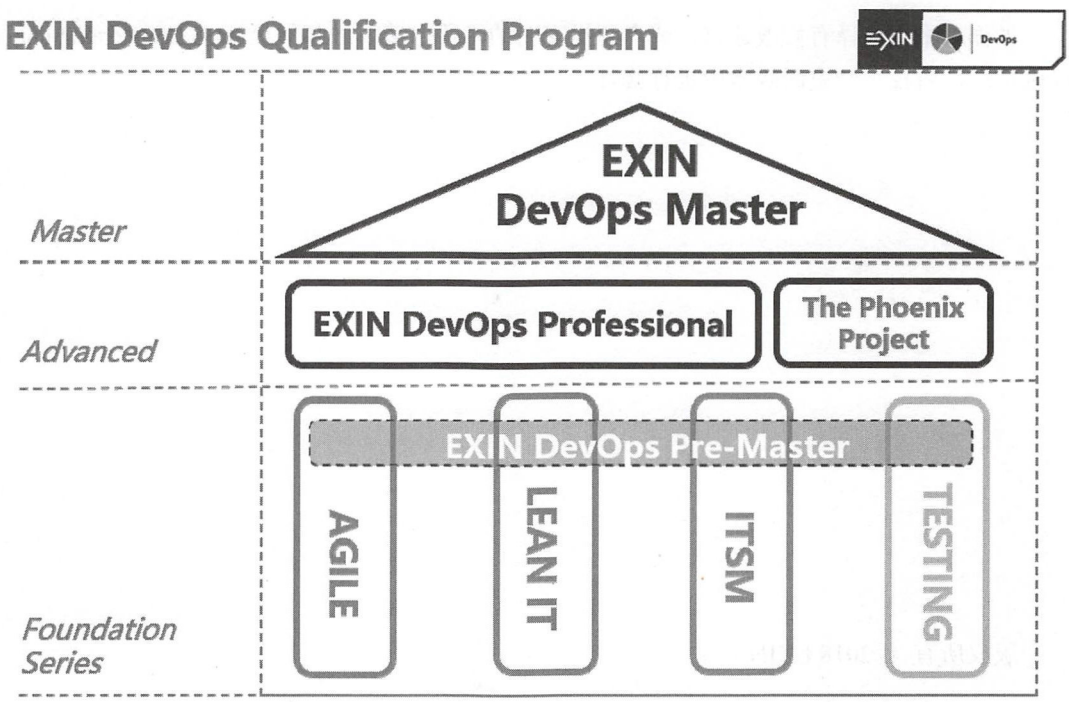






第一部分 EXIN DevOps 认证体系概览

1 DevOps 体系框架



EXIN DevOps 认证体系已搭建起完整的框架。请注意，EXIN DevOps 认证体系中各门认证均无前置认证要求，考生可根据自身情况选择，考取合适的认证。

特别致谢：

在本附录中，EXIN DevOps Master 基础术语表、样题、考试大纲及细则分别由以下几位老师共同审校及修订，特此声明，以表感谢。

以下按照姓的拼音首字母顺序排名：

基础术语表和样题审校：董伟、李伟、李岩、李毅、刘颋、马博文、孙翊威、汪珺、王磊、萧田国、许峰、张引

考试大纲及细则修订：卢梦纯

如你对术语翻译有修改建议，或者想提交新的术语，请发送邮件至 info.china@exin.com，经确认后，将在下一更新版本中进行修订。

EXIN

EXIN DevOps Master

认证备考指南 & 模拟题

EXIN DevOps

附录E　参考网站

Businessdictionary	[Businessdictionary]	http://www.businessdictionary.com
Collabnet	[CollabNet]	https://www.collab.net
dbmetrics	[dbmetrics]	http://www.dbmetrics.nl
Devops	[devops]	http://devops.com
EXIN	[Exin]	http://www.exin.nl
Gladwell	[GLADWELL]	http://www.gladwill.nl
IIR	[IIR]	http://www.IIR.nl
Investopedia	[Investopedia]	https://www.investopedia.com
ITMG	[ITMG]	http://www.ITMG.nl
ITPedia	[ITPEDIA]	http://www.itpedia.nl
Itrevolution	[itrevolution]	https://itrevolution.com
newrelic	[Newrelic]	http://newrelic.com/devops
What is	[What is]	http://whatis.techtarget.com
WIKI	[WIKI]	http://nl.wikipedia.org / wiki / Cloud_computing

	problem is	
	• Recommendations for how to proceed	• 建议：如何解决问题的建议
SCM	Software Configuration Management	软件配置管理
SIT	System Integration Test	系统集成测试
SLA	Service Level Agreement	服务级别协议
SMART	Specific， Measurable，Accountable，Realistic，Timely	具体的、可度量的、可实现的、有时限的
SMED	Single-Minute Exchange of Die	一分钟换模法
SNMP	Simple Network Management Protocol	简单网络管理协议
SPOC	Single Point of Contact	唯一联系点
SQL	Structured Query Language	结构化查询语言
SRG	Standard，Rules & Guidelines	标准、规则和指南
ST	System Test	系统测试
SVN	SubVersioN	开源的代码版本控制软件
TiCS	Tiobe Code Standard	Tiobe 代码标准
TSQL	Transact Structured Query Language	处理结构化查询语言（SQL 程序设计语言的增强版）
TTM	Time-To-Market	上市时间
UAT	User Acceptance Test	用户验收测试
UP	User Profile	用户配置
UT	Unit Test	单元测试
VCS	Version Control System	版本控制系统
WAN	Wide Area Network	广域网
WIP	Work in Progress	在制品
XP	eXtreme Programming	极限编程

LDAP	Lightweight Directory Access Protocol	轻量级目录访问协议
MRI	Minimum Required Information	最少需求信息
MST	Master Test Plan	主测试计划
MT	Management Team	管理团队
NFR	Non-Functional Requirement	非功能性需求
OLA	Operational Level Agreement	维护水平协议
P&O	Personnel & Organisation	个人与组织
PaaS	Platform as a Service	平台即服务
PAT	Production Acceptance Test	产品验收测试
PBA	Patterns of Business Activities	业务活动模式
PDCA	Plan，Do，Check，Act	计划、实施、检查、改进
PK	Primary Key	主键
PST	Performance Stress Test	性能压力测试
QA	Quality Assurance	质量保证
QC	Quality Control	质量控制
QCA	Quality Control & Assurance	质量控制和保证
RFC	Request for Change	变更申请
RMT	Release Management Team	发布管理团队
ROI	Return On Invest	投资收益
RUM	Real User Monitoring	真实用户监控
S-CI	Software Configuration Item	软件配置项
S-CMDB	Software Configuration Management Data Base	软件配置管理数据库
SaaS	Software as a Service	软件即服务
SACM	Service & Asset Configuration Management	服务与资产配置管理
SBAR	• Situational information to describe what is happening • Background information or context • An assessment of what they believe the	• 情况：描述发生了什么情况 • 背景：背景信息或上下文 • 评估：对大家认为的问题进行评估

DML	Definite Media Library	最终介质库
DML	Data Manipulation Language	数据操作语言
DMZ	Demilitarized Zone	隔离区
DocID	Document Identifier	文档标识
DOD	Definition of Done	完成的定义
DOR	Definition of Ready	就绪的定义
DTAP	Development-Test-Acceptance-Production	开发—测试—验收—生产
E2E	End-to-End	端到端
ESB	Enterprise Service Buss	企业服务总线
ETL	Extract Transform & Load	提取、转换和加载
EUX	End User Experience	终端用户体验
FAQ	Frequently Asked Questions	常见问题
FAT	Functional Acceptance Test	功能验收测试
FK	Foreign Key	外键
HRM	Human Resource Management	人力资源管理
IaaS	Infrastructure as a Service	基础设施即服务
IaC	Infrastructure as Code	基础设施即代码
IT	Information Technology	信息技术
ICT	Information Communication Technology	信息通信技术
ID	Identifier	标识
INVEST	Independent，Negotiable，Valuable，Estimable，Small，Testable	独立的、可商议的、有价值的、可估算的、小的、可测试的
ITIL	Information Technology Infrastructure Library	信息技术基础架构库
ITSM	Information Technology Service Management	信息技术服务管理
JKK	Ji-Kotei-Kanketsu	质量检查
KPI	Key Performance Indicator	关键性能指标
LAN	Local Area Network	局域网
LCM	Life Cycle Management	生命周期管理

附录 D　缩略词

API	Application Programming Interface	应用程序接口
ASL	Application Services Library	应用服务库
BiSL	Business information Services Library	商业信息服务库
CAB	Change Advisory Board	变更顾问委员会
CAMS	Culture，Automation，Measurement and Sharing	文化、自动化、测量和分享
CEMLI	Configuration，Extension，Modification，Localisation，Integration	配置、扩展、修改、本地化、集成
CI	Configuration Item	配置项
CFO	Chief Financial Officer	首席财务官
CIO	Cief Information Officer	首席信息官
CMDB	Configuration Management Data Base	配置管理数据库
CMS	Content Management System	内容管理系统
CPU	Central Processing Unit	中央处理器
CRM	Customer Relationship Management	客户关系管理
CSF	Critical Success Factor	关键成功因素
CT	Component Test	组件测试
CVS	Concurrent Versions System	开源的版本控制系统
DDL	Data Definition Language	数据定义语言
DevOps	Development Operations	开发即运维
DDL	Data Definition Language	数据定义语言

续表

术语	解释
周期时间（精益）	连续两个单元离开工作（leaving the work）或制造过程之间的平均时间
周期时间（流动时间）	周期时间衡量的是整个系统的完成率或工作能力，周期时间越短意味着浪费的时间越少，在需求已经完成时不会存在没有完成的工作和进程
准时制	准时制意味着建立一个流水线式的单件流供应链
作战室（Obeya）	Obeya 表示作战室，服务于两个目的： • 信息管理 • 现场决策

续表

术语	解释
验收测试用例	对于开发人员来说，验收测试用例给出了“我怎么知道我什么时候完成？”的答案。对于用户来说，验收测试用例给出了“我得到了我想要的吗？”的答案。验收测试用例包括功能验收测试（FAT）用例、用户验收测试（UAT）用例和产品验收测试（PAT）用例。FAT 和 UAT 应该用业务语言来表达
仪表盘	仪表盘（Kibana）显示许多保存的可视化图形
猿猴军团（可靠性监控）	猿猴军团由产生各种各样的故障、检测异常情况、测试生存能力的服务（猴子）组成。目标是保持云服务的可靠性、安全性和高可用性。目前在猿猴军团中有 3 只猴子： • 看门猴子（未使用的资源） • 混乱猴子（尝试关闭服务） • 一致性猴子（不符合规则）
约束理论	这是一种方法论，用于识别阻碍达成目标的最重要的限制因素，并且系统地改进这个约束，直到该约束不再是限制因素
运维场景	必须由运维团队实施的工作应该用运维故事书面定义，这样运维场景才能按优先级排序和管理
在制品限制	这是在看板流程中使用的关键绩效指标，用于最大化已启动但尚未完成的项目数。限制在制品数量是提高软件开发流水线通过能力的绝佳方法
制品	就是生产制造的一个产品。在 DevOps 中，提交阶段的输出物是二进制文件、报告和元数据，这些也被称为制品
制品库	制品的存储中心被称为制品库。制品库用于管理制品及其相互的依赖关系
质量检查	质量检查意味着 100%完成一项工作。这个方法意味着： • 对目标理解清晰 • 理解正确的工作方式 • 确保高质量地工作 • 使工作 100%正确地完成，不将缺陷传递到下游工序 • DOD 至关重要 • 在没有检查的情况下维持必需的质量要求

续表

术语	解释
	• 第三步是持续实践和学习
声明式编程	这是一种编程范式，它表达了计算逻辑而不描述其控制流程。 一个例子是数据库查询语言，如 TSQL 和 PSQL
提交阶段	这是部署流水线中的一个阶段，即将源代码编译为目标代码。这个阶段包括单元测试用例的性能
提前持续改善	提前持续改善比持续改善更进一步，不但自己的活动得到改善，而且在上游执行且影响下游的活动也得到改善，以这种方式建立改进整个系统的问题反馈环
微服务架构	此体系结构由一组服务组成，其中每个服务提供少量功能，系统的总功能来自同时在生产中组合多个版本的服务，并且可相对容易地回滚到以前的版本
微流水线	在极少数情况下，为了生成整个应用程序，需要多个部署流水线。这可以通过每个应用程序组件使用一个流水线来完成。然后，将所有这些组件集合在一个中心流水线中，并且使整个应用通过验收测试、非功能性测试，将整个应用程序部署到测试和生产环境中
文化、自动化、测量和分享（CAMS）	CAMS 是文化、自动化、测量和分享的缩写。 • 文化：文化与 DevOps 的人员和流程有关。没有好的文化，自动化尝试就会毫无结果 • 自动化：发布管理、配置管理及监控和控制工具应能实现自动化 • 测量：“如果你不能测量它，你就无法管理它”或“如果你不能测量它，你就不能改进它” • 分享：分享想法和问题的文化对帮助组织改进至关重要。分享要创建反馈回路
文化债	有 3 种债务形式：文化债、技术债和信息债。这种形式的债务是指在组织结构、雇佣策略、价值观等方面存在决策缺陷。这些债务将产生利息，并且会导致 DevOps 团队成熟度较低。文化债可以通过是否存在孤岛、工作流程限制、沟通不畅、浪费等来确认
协作	协作是 DevOps 的四大支柱之一。协作是指 DevOps 团队的每个人共同努力以实现共同目标的一种协同工作方式。这种协作有如下表达形式：一对一编程、展示每周的进展、文档等

续表

术语	解释
蓝绿部署	以使用其他系统。这减少了停机时间的风险，因为切换时间可能不到一秒钟
浪费	浪费包括在制造过程中不为客户增加价值的活动。在 DevOps 中的例子是:不必要的软件特性、沟通延迟的应用系统响应时间、难以忍受的官僚流程
浪费（Muda）	Muda 是一个日语词汇，代表浪费，与生产系统有关
免责	这种方法强调学习而不是惩罚。在 DevOps 中，这是从错误中学习的基本思想之一。DevOps 团队的精力应该用在从错误中学习，而不是责备某人
命令式编程	这是一种编程范式，它使用语句来改变程序的状态。命令式编程着重于程序如何运作，以便成为计算机可执行的命令。例子有 COBOL，C ++，BASIC 等。 与声明式编程相比，这个术语经常被使用，它侧重于程序应该完成什么，而不指定程序应该如何实现结果
配置管理	配置管理是指对所有制品及其相互关系进行存储、检索、唯一标识和修改的过程
前置时间	前置时间是指从提出请求到递交最终结果或从客户的角度来看完成请求事件的时间
亲和力	DevOps 强调协作和亲和力。协作关注 DevOps 团队中个人之间的关系，亲和力则更进了一步。亲和力强调 DevOps 团队通过学习分享故事和互相学习，在不同人群间建立共享的组织目标和同理心
轻量型 ITSM	ITSM 的这种变体严格地专注于带有最低需求信息的业务连续性。每个组织的最少需求信息（MRI）依赖于它的业务
容器	容器是一种独立的架构，开发人员利用它可以独立于底层操作系统或硬件去构建应用程序。开发人员可以使用容器中的接口完成（对底层资源的调用）。我们可以部署完整的容器，而无须在环境中单独安装应用程序，这样可以精简大量的依赖关系并防止发生配置错误
三步工作法	三步工作法是在《凤凰项目》中介绍的，三步工作法是一个用于设计 DevOps 的流程、程序和实践及规定步骤的有效方法。 • 第一步是流，理解和增大工作流（从左到右） • 第二步是反馈，建立短的反馈回路，促进持续改善

续表

术语	解释
基础设施即代码(IaC)	基础设施即代码（IaC）可以对这些基础设施组件的设置进行编程，并且通过使用机器可读的定义文件在流水线上部署这些设施，而不是利用物理硬件配置或交互式配置工具去部署它
价值流映射	价值流映射是一个精益工具，它描述了信息、材料和跨功能条块工作的流程，重点是量化浪费，包括时间和质量
金丝雀发布	金丝雀发布是一次只发布给一小部分用户。如果这个小范围使用的版本可以正常工作，则可以将该版本部署到所有用户。金丝雀（发布）这个词指的是一种古老的传统，即让金丝雀探测煤矿里是否有毒气
精益工具	• A3 思维（问题解决） • 连续流（消除浪费） • 持续改善 • 看板 • 关键绩效指标（KPI） • PDCA • 根本原因分析 • SMART 原则 • 价值流映射（描绘流） • 准时制（没有缺陷传递到下游工序）
看板	这是需要某物时发出信号的系统。看板是一个管理物流生产链的系统，是 Taiichi Ohno 在丰田公司开发的用于找到一种尽可能达到高水平生产的系统。看板常常用于管理，特征之一就是无库存原料的拉式驱动生产，可用于实施准时制生产（JIT）
快乐路径	应用程序通过接收、编辑、存储和提供信息来支持业务流程。执行信息处理的步骤称为快乐路径。以可替代方式给出的步骤称为替代路径。在这种情况下，通过另一个导航路径将能实现相同的结果。导致错误的应用程序爬网称为错误路径
蓝绿部署	蓝色和绿色是指两个相同的生产系统。其中一个用于部署最终的新版本。如果验收成功，那么这个环境就成为新的生产环境。如果生产系统出现故障，则可

续表

术语	解释
错误路径	参考快乐路径
单件流	精益方法意味着具备快节奏、平滑流能力的 DevOps 团队一次只为一件事工作，这也是适用于 Gene Kim 提出的三步工作法的第一种方法
独立的、可商议的、有价值的、可估算的、小的、可测试的（INVEST）	• 独立的：产品待办事项应该是独立的，某事项对其他产品待办事项没有内在依赖性 • 可商议的：产品代办事项直到它们成为迭代的一部分前，总是可以更改、重写甚至丢弃的 • 有价值的：产品待办事项必须为利益相关者提供价值 • 可估算的：产品待办事项的大小必须始终可以估算 • 小的：产品待办事项不应该大到无法为其制订确定的计划/任务/（排定）优先级 • 可测试的：产品待办事项或其相关说明必须提供必要的信息，以便使测试开发成为可能
二进制代码	编译器将源代码转换为目标代码。 目标代码也被称为二进制代码。源代码对于人类来说是可读的，但是目标代码只能用于计算机，因为它们是用十六进制编写的
反模式	反模式是对模式错误理解与解释的一个例子，通常用于解释模式的价值
构建异常	由于应用程序源代码中的错误而导致构建失败
行为驱动开发	软件开发要求用户定义功能性和非功能性需求。行为驱动开发就是基于这一概念的。不同的是，这些需求的验收标准应该写明客户对应用程序的行为的期望。验收标准可以按照“Given - When - Then”的格式来编写
行走骨架	行走骨架意味着做尽可能少的工作，使得所有关键要素就位
基础设施管理	为了支持运行在基础设施上的应用能够正确工作，基础设施管理由所有基础设施产品和服务的生命周期管理组成
基础设施即代码（IaC）	通常基础设施组件必须进行配置，以便达成所请求的功能和质量，例如，防火墙的规则集或允许访问的网络 IP 地址。这些配置通常存储在配置文件中，使运营团队能够管理基础设施组件的功能和质量

续表

术语	解释
SBAR（情境、背景、评估、建议）	这项技术提供了指导，以确保关注或评论是以富有成效的方式表达的。在这种情况下，关注它的人们必须提供以下信息： • 描述事情正在发生的情境 • 背景信息或上下文 • 对他们认为的问题的评估 • 给出如何推进的建议
替代路径或备用路径	参考快乐路径
编程范式	一种构建计算机编程语言的结构和要素的风格或编程方式
测试驱动开发	测试驱动开发是在定义测试用例并执行完成后编写源代码的方法。直到测试用例条件满足，才编写并调整源代码
测试套件	被构建用于促进集成测试的软件。其中，测试桩通常是正在开发中的应用程序的组件，在应用程序开发后由可工作组件替换（自上而下集成测试），测试套件在测试应用程序的外部，并且模拟在测试环境中不可用的服务或功能
持续改善	Kaizen 是日语“改善”的意思，Kaizen 用于改善生产系统。Kaizen 的目标有： • 消除浪费 • 准时制生产 • 产品标准化 • 持续改进周期 持续改善意味着按周、日循环进行 PDCA，可以通过问 5 次“为什么”来找到失败的根本原因。要遵循以下步骤： • 用数据来定义问题 • 确保每个人清晰地认识问题 • 设置一个发现问题的假设 • 定义防范措施去验证假设 • 基于日常活动定义防范措施 • 需要测量每周的 KPI，这样人们可以有成就感

附录 C　术语表

术语	解释
5S	源自日本的秩序与清洁原则。与这些日语词汇相对应的英文词汇是： Seiri（整理）：Sort（整理） Seiton（整顿）：Arrange（整顿） Seisō（清掃）：Cleaning（清扫） Seiketsu（清潔）：Standardize（清洁） Shitsuke（躾）：Hold or Systematize（素养） [WIKI]
A/B 测试	A/B 测试意味着两个不同版本的应用程序或网页被投入生产环境，看哪个性能更好。可以使用金丝雀发布，但也有用其他方法来执行 A/B 测试的
Given-When-Then	Given-When-Then 是一种定义验收标准的格式。它帮助利益相关方理解功能实际上是如何工作的。Given——事实是……When——当我这样做……Then——会发生这种情况
Kata	Kata 是一种结构化的思维和行动方法（行为模式），通过实践将行为模式变成本能。可以通过 4 步养成本能： • 方向（目标） • 现状（IST 形势） • 目标状况（SOLL 形势） • PDCA（戴明环） 从架构的角度来看，迁移路径也应该被加入 Kata 中。迁移路径显示了实现 SOLL 形势的方法

Story 故事

System Test (ST) 系统测试

Tag 标签

Technical debt 技术债

Technical design 技术设计

Test case 测试用例

Test driver 测试驱动程序

Test stub 测试桩

Test types 测试类型

Theme 主题

Tool integration 工具集成

Tooling portfolio 工具组合

Traceability 可追溯性

Trunk 主干

Unit Test (UT) 单元测试

User Acceptance Test (UAT) 用户验收测试

User story 用户故事

Version 版本

Version Control System (VCS) 版本控制系统

Version management 版本管理

Version number 版本号

Version strategy 版本策略

Vertical feature split 垂直特性分割

Waterfall 瀑布式

Waterfall project 瀑布式项目

False positive 漏报

Feature 特性

Feature branch 特性分支

Feature splitting 特性分割

Flow 流

Forward releasing 前向发布

Functional Acceptance Test (FAT) 功能验收测试

Functional decomposition 功能分解

Functional design 功能设计

Funnel 漏斗

Health model 健康模型

Horizontal feature split 水平特性分割

Impact 影响

Increment 增量

Information debt 信息债

Integration Test (IT) 集成测试

Iterative 迭代

Key Performance Indicator (KPI) 关键绩效指标

Life cycle management 生命周期管理

Make or buy 自制或外购

Master Test Plan (MTP) 主测试计划

Merge 合并

Meta data 元数据

Monitoring layer model 监控层模型

One piece flow 单件流

Pattern 模式

Pattern language 模式语言

Performance Stress Test (PST) 性能压力测试

Primary key / Foreign key 主键/外键

Process blue print 流程蓝图

Production Acceptance Test (PAT) 产品验收测试

Push and pull 推送和拉动

Quality Control & Assurance (QCA) 质量控制与保证

Release branch 发布分支

Release planning 发布计划

Repository 存储库

Requirement 需求

Risk 风险

Risk-based acceptance 基于风险的接受

Roadmap 路线图

S-CMDB Baseline 软件配置管理数据库基线

Service 服务

Service & Asset Configuration Management (SACM) 服务与资产配置管理

Service desk 服务台

SLA 服务级别协议

Software Configuration Item (S-CI) 软件配置项

Software Configuration Management Data Base (S-CMDB) 软件配置管理数据库

Software Configuration Management（SCM） 软件配置管理

Standard, Rules and Guidelines（SRG） 标准、规则和指南

Static requirement 静态需求

附录 B　词汇表

Acceptance criterion　验收标准

Agile development　敏捷开发

Agile processes　敏捷流程

Agile project　敏捷项目

Agile Scrum　敏捷 Scrum

Annotation　注解

Architecture model　结构模型

Baseline　基线

Best of breed/Integrated tooling　单项优势/集成工具

Branching　分支

Branching strategy　分支策略

Business case　商业论证

Chain test　链测试

Change authorities　变更授权

Change control　变更控制

Check-in　签入

Check-out　签出

CMDB-baseline　CMDB 基线

Component Test (CT)　组件测试

Configuration Item(CI)　配置项

Configuration Management Data Base (CMDB)　配置管理数据库

Continuous Delivery (CD)　持续交付

Continuous Integration (CI)　持续集成

Continuous monitoring　持续监控

Continuous testing　持续测试

Data modeling　数据模型

Demarcation　界限

Deployment pipeline　部署流水线

Definition of Done (DOD)　完成的定义

Definition of Ready (DOR)　就绪的定义

Dynamic requirement　动态需求

Epic　史诗

Event black list　事态黑名单

Event catalogue　事态目录

Event management　事态管理

Event white list　事态白名单

Exception　异常

Exception management　异常管理

False negative　误报

Enterprise DevOps Koichiro" "White Paper", 2016.

Kim 2014 Gene Kim, Kevin Behr, George Spafford "The Phoenix Project", IT Revolution Press, ISBN-13: 978 09 88262 508, 2014.

Kim 2016 Gene Kim, Jez Humble "The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations, Patrick Debois, John Willis", IT Revolution Press, ISBN-13: 978 19 42788 003, 2016.

附录 A　参考资料

文献	出版物
Best 2014a	B. de Best, “Acceptatiecriteria”, Dutch language, Leonon Media, ISBN 13: 978 90 71501 784, 2014.
Best 2015a	B. de Best, “Agile Service Management met Scrum in de Praktijk”, Dutch language, Leonon Media, ISBN13: 978 90 71501 845, 2015.
Best 2015b	B. de Best, “Ketenbeheer in de Praktijk”, Dutch language, Leonon Media, ISBN13: 978 90 71501 852, 2015.
Best 2017a	B. de Best, “Cloud SLA”, English language, Leonon Media, ISBN13: 978 94 92618 009, 2017.
Best 2017b	B. de Best, “SLA Templates”, English language, Leonon Media, ISBN13: 978 94 92618 030, 2017.
Best 2018	B. de Best, “Agile Service Management with Scrum”, English language, Leonon Media, ISBN13: 978 94 92618 085, 2018.
Davis 2016	Jennifer Davis, Katherine Daniels, “Effective DevOps: Building a Culture of Collaboration, Affinity, and Tooling at Scale”, O'Reilly Media; 1 edition, ISBN-13: 978 14 91926 307, 2016.
Humble 2010	Jez Humble, David Farley “Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation”, Addison-Wesley Professional; 1 edition, ISBN-13: 978 03 21601 919, 2010.
Toda 2016	(Luke) Toda, President Strategic Staff Services Corporation and Director of TPS Certificate Institution Nobuyuki Mitsui, CTO of Strategic Staff Services Corporation, “Success with

监控

监控工作是模拟沙盘内容的一部分，团队的工作需要被 CFO 和模拟沙盘的领导者监控。CFO 必须关注执行任务过程中达成的收益和损失。此外，股票价格必须通过凤凰项目沙盘演练得到提升。

32.4.3 什么时候用这套模拟沙盘

这套模拟沙盘应该被用在重组组织的工作中。参与者将很愉快地参与沙盘游戏并从中学会怎样和别人一起工作，这是组织转型阶段的一个重要成功因素。

规划

模拟沙盘中最重要的一点就是规划。在图 32-2 中可以看到看板。凤凰项目的工作必须被规划。看板解释了怎样帮助人们执行任务。4 种类型的工作是 Gene Kim 认可的：商业项目、ICT（信息通信技术）项目、变更和计划外工作，沙盘的内容包含了这 4 种工作。三步工作法是和 DevOps 流程计划阶段紧密相关的。

编码

模拟沙盘的内容并不是像编程那样的技术工作，模拟沙盘适合 DevOps 团队的所有角色，更重要的是可以学习协作的有效性和亲和力的重要性。所有 DevOps 的效果都包含在模拟沙盘中。

构建

由于没有编程，所以也不需要构建，虽然交付的软件必须遵从基础设施和中间件的正确配置，这是模拟沙盘内容的一部分。

测试

测试是模拟沙盘中规划之后最重要的内容之一。没有测试反馈的 DevOps 团队，效能是不会提高的，应用测试甚至可以驱动开发。

发布

模拟沙盘中包含变更管理员的角色，该角色负责发布被批准的变更。然而，模拟沙盘的领导者是最终的裁决人，他能撤销或推翻变更管理员做出的错误决定。

部署

如果系统没有被构建，那么部署也就不会执行。但是，配置管理是模拟沙盘内容的一部分，这意味着必须在 4 个回合中执行重要的检查，以跟踪配置管理信息。

运维

运维是模拟沙盘内容的一部分。在 4 轮比赛中事故的发生会影响团队的生产效率。

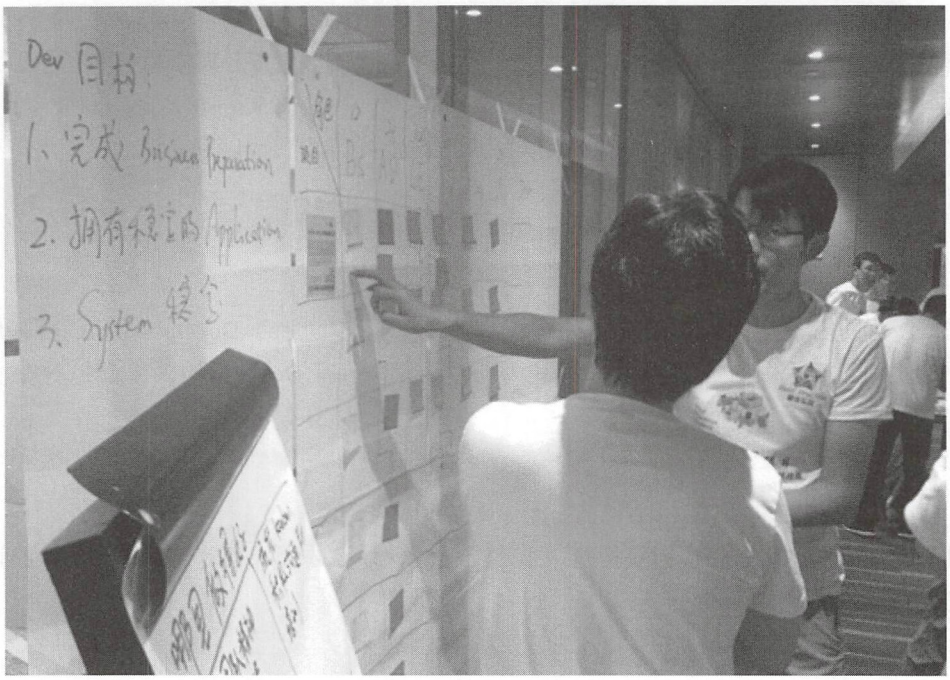


图 32-2 制订一个好的规计划是模拟沙盘中非常重要的成功要素

32.4.2 这和 DevOps 流程是怎样联系在一起的

在图 32-1 中，开发团队用深蓝色来代表（图中深色桌签），运维团队用浅蓝色来代表（图中浅色桌签）。两个团队需要协同工作，一起为客户服务。DevOps 的流程在文章#02 中做了说明。

DevOps 流程（见图 32-3）与凤凰项目沙盘的关系如下。

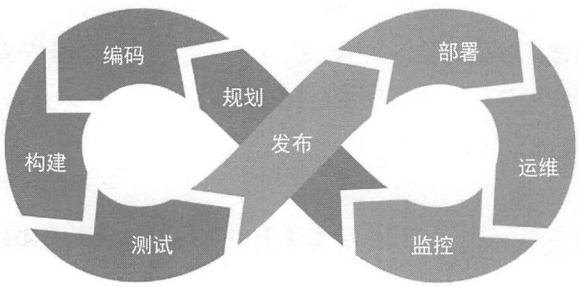


图 32-3 DevOps 流程的各阶段

32.4.1 模拟沙盘是如何工作的

凤凰项目模拟沙盘是由一个约 8～11 人组成的团队共同完成的。参与者在一天内需要完成 4 轮模拟。每轮游戏时间约 30 分钟。剩下的时间内，团队需要总结及思考如何提高团队绩效。事实上，模拟沙盘过程中的经验教训都是围绕如何运用 Gene Kim 所提出的三步工作法的。

模拟沙盘开场由 Parts Unlimited 公司 CIO/CTO Steve 宣布公司股票价格下跌，接二连三地在杂志上有负面报道，并且公司过往的经验已经不可行了。

4 轮模拟沙盘完成后，公司当前呈现的负面趋势必须扭转成正面向上的趋势。

这个艰巨的任务必须由 Parts Unlimited 公司凤凰项目团队来完成。每个团队成员会被分派到一个或多个在《凤凰项目》一书中介绍的角色，每个角色都有相应的岗位描述。模拟沙盘中会有一些预先定义好的团队。

在 4 轮模拟过程中会用到各种卡片，这些卡片会指出需要完成哪些工作。其中最重要的一点是只有团队紧密合作才能完成更多的任务，取得更好的成绩。成功的团队都需要在每轮中评估当前的形势，做出最优的决策。在一天的最后，成绩会以可视化的图表呈现，经验教训总结在整个过程中也会被反复提及。图 32-1 和图 32-2 是模拟沙盘过程中的一些场景。

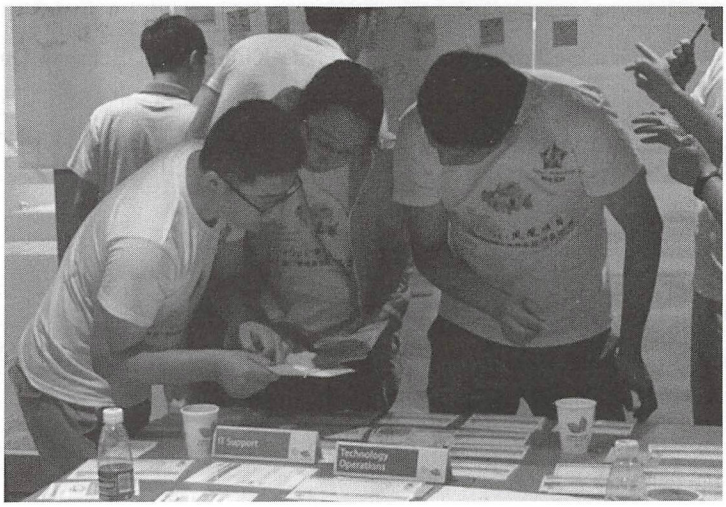


图 32-1 在模拟沙盘中团队成员协同工作，实现项目交付

格及最好的质量。

32.3 概念

三步工作法

Gene Kim 定义了使 DevOps 可工作的方法，共包含 3 个步骤，各步骤需要依次执行。这 3 个步骤是：

- 第一步，系统地思考
  - 关注整体，不只是优化某个局部、某个团队的效率。
  - 聚焦价值流（看板）。
  - 增加流动，减少浪费（精益生产）。
- 第二步，增强反馈
  - 缩短反馈环，优化流程。
- 第三步，培养持续实践及学习的文化
  - 从失败中学习，承担风险。
  - 达到精通需要反复实践。

32.4 最佳实践

凤凰项目沙盘是由 GamingWorks.nl 公司基于 Gene Kim, Keven Behr 和 George Spafford 撰写的小说《凤凰项目》开发的。在这个小说中，讲述了 Parts Unlimited 公司如何通过 Gene Kim 定义的三步工作法着手开始 DevOps 旅程。书中所有的场景都贴近现实，并且可以在众多组织中找到原型。

模拟沙盘可以有以下几个用途：

- 研究什么是 DevOps，它可以给我们带来什么，它的附加值是什么。
- 建立 DevOps 的意识，并且同时获得管理层、业务层及所有员工的认可。
- 触发一个新的行动路线或组建一支新的团队开始 DevOps 的旅程。
- 提高团队协作、沟通、决策、领导力，以及使用看板等的能力。
- 作为理论培训的补充。

32

凤凰项目沙盘（#31）

32.1 引言

本文将介绍如何开始使用 DevOps。我接触过许多使用 DevOps 的组织，它们的方法可能各不相同，但都是从培养员工、团队人员的意识开始的。想要 DevOps 的旅程获得成功，培养 DevOps 意识这个环节是非常重要的。本文通过“凤凰项目沙盘”这个实践来总结如何建立 DevOps 意识，简略地介绍在文章#02 中提到的与 DevOps 相关的流程。

32.2 术语

精益

精益原则起源于日本制造业。John Krafcik 在 1988 年发表的 *Triumph of the Lean Production System* 文章中第一次使用了该术语。减少浪费可以提高生产率并节约成本。另外，精益也是一整套协助识别并持续稳定消除浪费的工具。以下是部分精益工具：SMED、价值流映射（Value Stream Mapping）、5S、看板、Poka-Yoke、高效维护（Total Productive Maintenance）、时间批次消除（Elimination of Time Batching）、混杂模式流程（Mixed Model Processing）、等级排序聚类（Rank Order Clustering）、单点计划（Single Point Scheduling）、单元重构（Redesigning Working Cells）、多流程控制、控制图。

流

在 DevOps 中经常会使用精益原则，其中最重要的一个应用是流。DevOps 团队需要在一个特定的没有等待时间、没有备件的节奏下交付产品，因此，能被市场长时间认可的是最低的价

场也将增长。目前，IaaS 和 PaaS 服务确实接管了 DevOps 团队的某些角色，只剩下一些配置工作需要 DevOps 团队成员来进行，但云供应商也可能会提供基本的业务解决方案，或者替代现在由 DevOps 团队提供的某些功能。

3. 信息管理

在基础设施管理和应用管理不断减少的情况下，信息管理部分将受到更多关注。由于信息质量是企业服务的关键，因此信息质量将成为人们关注的焦点。服务管理的未来是关于信息管理的，而不是关于应用程序管理和基础设施管理的。

31.4.4 DevOps 是个骗局吗

逆转 DevOps 意味着回到哪里？

你能拆散部署流水线从而获得时间和金钱吗？

没有协作和亲和力，你能获得利润吗？

答案是否定的。

在过去的 40 年间，许多方法、模型和框架不断产生、实施和淘汰，但是 DevOps 并不是一种方法或模型，而是一个符合当前业务需求的行动（movement）。毫无疑问，世界将发生变化，企业的运作方式也会发生变化。然而，对 DevOps 的投入将持续相当长的时间，比过去几十年的所有举措都将持续更久。

TTM 的延迟。

3. 忽视信息管理

团队中缺少信息管理知识意味着反馈速度较慢，亲和力也较小。其风险在于对业务需求错误理解，这会导致时间、金钱和质量的损失。

4. 受技术债、组织债和信息债的拖累

“支付”技术、组织、信息债的“利息”可能为 DevOps 商业论证带来损失。技术债必须通过软件重构来偿还，重构旨在消除软件缺陷。

这会花费大量的金钱和时间。组织债必须通过组织重组来偿还，会影响 DevOps 团队的速度。初期的 DevOps 团队容易忽略维护正确的元数据，偿还信息债（见文章#13）要求重构遗留系统。这些债务需要偿还，而且拖的时间越长，它们的纠正成本就越高。因此，短期收益很快被人淡忘，预期的投资回报率也将无法实现，必须进行额外的投资来偿还这些债务。

5. 组织失去亲和力

成功的 DevOps 意味着实现业务目标。协作通常被视为充分的业务聚焦，与不同学科背景的人一起协同工作是其中的关键。否则，组织可能遭受时间和金钱的损失。

31.4.3 未来发展

未来无法预测，但趋势有迹可循。市场上可以看到以下发展趋势。

1. 基础设施管理

在过去的 10 年中，许多组织的基础设施员工数量下降了 80%。其原因是云服务（IaaS，PaaS，SaaS）和虚拟化，这使得基础设施即代码（Infrastructure as Code，IaC）成为可能。就连政府机构也在（部分）转向云。基础设施管理的作用在不断减弱。

有些人担心未来的 Dev 和 Ops 会再次分道扬镳。然而，由于基础设施管理工作日益减少，而且 Dev 正在越来越多地接管基础设施管理的角色，这种情况不会发生。

2. 应用程序管理

越来越多的组织正在使用可配置的标准软件包，这种情况将一直持续。此外，SaaS 服务市

- 反馈更快减少了在错误的解决方案上的投入。
- 故障更少带来更好的用户印象、更高的客户满意度和更少的损失。
- 受控意味着风险得到管理和控制，最终降低成本。
- 更快的上市时间意味着更高的利润。

进行商业论证还意味着必须考虑实施 DevOps 的风险。在引入 DevOps 时，需要管理的重要风险是：

- 转型策略错误。
- 没有将 Ops 实际整合到开发、测试和部署的全过程中。
- 忽视信息管理。
- 受技术债、组织债和信息债的拖累。
- 组织失去亲和力。

1. 转型策略错误

重塑组织有许多策略。图 31-1 描述了一种简单的方法。哪种方法最适合组织的文化？每种方法都有各自的好处和风险，当然这些方法也可以组合使用。转型策略选择错误的代价是巨大的，因为完全的转型可能延迟甚至彻底失败。

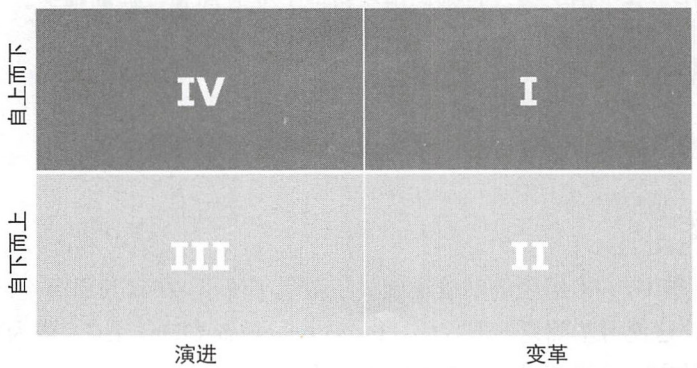


图 31-1 转型策略

2. 没有将 Ops 实际整合到开发、测试和部署的全过程中

许多组织选择仍然保持着两个团队（Dev 和 Ops），并通过一种矩阵的方式将它们组合在一起。Ops 与 Dev 团队通过矩阵的形式组合到一起是要花一些时间来磨合的。这样做的风险在于难以真正强有力地协作和建立较大的亲和力，共享的知识也有限，会造成额外的维护成本和

信息管理和基础设施专家的反馈对开发和调整业务服务提供很大的帮助；最后，一方面客户能更快地消费所交付的服务，另一方面他们的反馈也会得到更快的响应。

7. 故障更少

有了快速反馈，部署流水线中的失败情况就能被更早地检测到。而且，由于可以自动回滚到以前的版本，对于仅在生产环境中才出现的故障，其停机时间也更短。此外，DevOps 中可采用金丝雀发布，即只有少数用户可使用新服务，万一失败，影响面会很小。

8. 受控

敏捷 Scrum 专注于软件产品的开发，而 DevOps 则专注于交付可在生产环境中运行的服务。DevOps 的 Ops 是关于控制的。许多敏捷 Scrum 团队使得 Dev 和 Ops 之间的关系紧张，而 DevOps 的方法则明确地整合了双方。因此，DevOps 不像敏捷 Scrum 那样，Dev 只是使劲地推动 Ops 赶快交付，而是 Dev 和 Ops 两个团队共同携手开发和交付，收获的不仅是更好的亲和力，也很好地将 Ops 的控制措施融入开发过程中。在许多敏捷 Scrum 团队中，ITIL®、ASL®和 BISL®的控制被忽略了，而在 DevOps 中，我们再次看到了其在应对风险时的应用。

9. 更快的上市时间

Business DevOps 的方法（业务由 DevOps 团队中的信息管理人员提出）可以很好地推动 TTM。如今，组织甚至开始将 DevOps 团队与业务团队整合起来，IT 部门被关闭，IT 人员被整合到业务团队中。业务团队必须加快速度，才能在市场竞争中获胜，而 DevOps 方法具有实现这一目标的优势。

10. 成本更低

使用 DevOps 方法生产新的服务或调整服务所需的成本要低得多。它是结合以下 9 项好处来实现的：

- 协作能使人更好地理解需求、错误和对“英雄人物”的依赖，从而节省时间和金钱。
- 亲和力有助于形成更高效的组织，组织中的成员更好地合作，从而更有效地实现目标，使组织产生更多的利润。
- 可伸缩性有利于支撑更大的交易量。
- 自动化降低了劳动力成本。
- 部署更快缩短了 TTM，从而提高了利润。

的特定目标。戴明建议“打破部门之间的隔阂”，要落实这一建议，一个团队中需要拥有应用软件服务的完整生命周期的技能。将所有团队组合在一起共同为客户提供一个或多个服务组合，这样更容易实现服务的规范化和业务目标。

3. 可伸缩性

对 ICT 服务的可伸缩性需求非常旺盛。这不仅包括要实现软件的可伸缩性，还包括基础设施的可伸缩性。通过更好地解决 ICT 服务的技术问题，DevOps 可以比敏捷 Scrum 更好地实现可伸缩性。运维是开发的一部分，它们共同定义了服务的总体结构。

4. 自动化

DevOps 的基础是部署流水线。该流水线是通过使用精益原则来管理的，即从流水线的开始（需求）到最后（部署）的全过程都尽量减少浪费。流水线的许多部分是自动化的，以提升交付速度。在敏捷 Scrum 各团队中，他们通常各自选择最好的解决方案，而这些解决方案并不能有效协同。此外，许多手工的方法，如 Scrum 看板，也都是手工维护的。然而，DevOps 的基本原则之一是，通过尽可能多地建立自动化部署流水线，使其尽可能平稳地运行。

5. 部署更快

敏捷 Scrum 过程产生潜在的可交付产品，而 DevOps 的 DOD 关注的是一个可运行的服务。DevOps 的生命周期总是以部署为结束的，这种部署比敏捷 Scrum 方法要快得多。敏捷 Scrum 方法可能在许多冲刺中无法及时发布支持服务运行的软件。例如，某个组织依赖企业服务总线（Enterprise Service Bus，ESB），该总线只能每年发布两次。这导致该组织开发的应用软件尽管每月能完成一个冲刺，却也只能与 ESB 的发布周期一样每年仅仅发布两次。

在 DevOps 中，这是一种不可接受的工作方式，因为“完成”才是铁律。

6. 反馈更快

反馈意味着所产生服务的任何不好的信息都会被返回给源头。

在 DevOps 中，这是通过使用“持续一切”（持续监控、持续测试、持续集成、持续部署/交付）来实现的。

这里最重要的是持续集成。开发人员将代码提交到集成环境之前，会事先将其在自己的本地环境签入预集成和本地测试；另外，还有持续测试，即在构建后执行自测和回归测试；此外，

31.4 最佳实践

下面将首先列出并论述 DevOps 的好处，其次将其与敏捷 Scrum 进行比较，最后展望 DevOps 未来的发展。本文不涉及其他敏捷方法，之所以选用敏捷 Scrum，是因为其应用广泛，即便在 DevOps 团队中也是如此。

31.4.1 DevOps 的优势

- 协作
- 亲和力
- 可伸缩性
- 自动化
- 部署更快
- 反馈更快
- 故障更少
- 受控
- 更快的上市时间（Time to Market）
- 成本更低

31.4.2 DevOps 和敏捷 Scrum 的对比

敏捷 Scrum 可以用在 DevOps 的开发过程中，但 DevOps 不仅仅是敏捷 Scrum。以下是 DevOps 与敏捷 Scrum 相比的几点优势。

1. 协作

在 DevOps 团队中开展工作意味着开发人员与运维人员有更多的接触。他们互相学习，一起工作。在业务 DevOps 团队中，信息管理员也加入了 DevOps 团队，这带来了更多的价值。因此，敏捷 Scrum 团队的协作是在一个团队中融合开发技能，DevOps 则是将服务的整个生命周期的所有技能融合到一个团队（信息、应用程序和基础设施）中。

2. 亲和力

亲和力就是多个团队一起为同一个业务目标而努力，而每个提供服务的团队也可以有自己

31

商业论证（#30）

31.1 引言

许多组织已经在使用敏捷 Scrum，并且怀疑 DevOps 是否真的能带来价值。其他组织对此仍抱有疑虑：下一步该怎么做？DevOps 是个骗局吗？本文通过使用 DevOps 的商业论证和对未来的预测来回答这些问题。

31.2 术语

商业论证

商业论证是一种定义启动一项新计划的收益和成本的方式。由于应对风险的策略需要时间和金钱的付出，所以务必考虑实施会遭遇的风险。

31.3 概念

敏捷 Scrum

《Scrum 指南》中定义了敏捷 Scrum 的概念。敏捷 Scrum 由 3 个角色组成，分别是产品负责人（product owner）、Scrum Master 和开发团队。其特点是时间盒（time boxing）及与业务密切的互动。敏捷 Scrum 的活动包括冲刺计划、每日站会、冲刺执行、评审和回顾，就绪的定义（Definition of Ready，DOR）和完成的定义（Definition of Done，DOD）起到质量控制的作用，最终输出潜在的可交付产品。

30.4.3 监控方法

- 基于关卡（tollgates）的监控
  - DOR。
  - DOD。
  - 升级（promotion）。
  - 辐射器（radiator）。
- 基于 SLA 规范的监控
  - DTAP 过程中的非功能性需求监控。
  - 前置时间（lead time）报告。
- 基于管理工具的监控
  - 缺陷数量。
  - 用户故事数量。
- 基于自动化的监控
  - 回归测试工具（缺陷）。
  - Tiobe 的质量标签（偏差标准、规则、指南）。
- 基于测量的监控
  - 性能监控。
  - 容量使用监控。
  - 可用性监控。

30.4.4 监控工具

市场上有大量的监控工具可以用来进行持续监控。

例如：

- Splunk
- Gensys (SPS)
- Nagios
- SCOM
- 等等

30.4 最佳实践

本文从持续监控的最佳实践开始，逐步描述持续监控的内容。

30.4.1 监控的最佳实践

- 尽早在 DTAP（开发—测试—验收—生产）过程启动监控，而不仅仅在生产环境中。
- 根据在生产中使用的监控设备，尽早发现缺陷。
- 不只监控软件，也要监控 DevOps 团队。

30.4.2 DevOps 每个阶段的 KPI 监控

DevOps 各阶段应监控：

- 开发过程
  - 速率。
  - TTM。
  - 逸出的缺陷。
  - 能力。
- 持续集成
  - 手工任务的数量和占比。
  - 没有通过交付流水线的次数（drop in delivery pipeline）。
- 持续测试
  - （重复出现的）缺陷数量。
  - 覆盖率。
  - 手工测试百分比。
  - 交付流水线每个阶段发生的缺陷。
  - UT, ST, SIT, pre-FAT, pre-UAT, pre-PAT, FAT, UAT PAT, PST in the DTAP。
- 持续交付
  - 在生产环境中但不在最终介质库（Definite Media Library，DML）中对象的数目。
  - 逸出的缺陷数量。

30

持续监控（#29）

30.1 引言

监控已经不是一个陌生的话题了，但持续监控向前又前进了一步。本文将介绍持续监控为我们带来了什么。

30.2 术语

绩效指标（KPI）

KPI 是用来衡量关键成功因素（CSF）的。CSF 是一种应对目标无法实现的风险的策略。

30.3 概念

持续监控

持续监控是 DevOps 的一个重要组成部分，它不仅监控软件（资源），还监控开发人员（人员）和开发过程（方法）。在所有环境中的资源都需要被持续测量，以便尽早发现错误。根据能力发展（知识、技能和态度）来衡量人员，方法包括测量速度（处理能力）和效率。

监控层模型

部署流水线和生产过程的监控有大量的监控工具可以选择，文章#6 概述了可用于监控服务的各种监控功能。

— *根据服务标准或性能范围进行测试。*

— *确定 DOD。*

- Ops
  — 确保硬件性能，如 CPU、网络带宽；确保应用程序性能，如应用程序的消息队列、消息、互联网对话、报表性能。

DevOps 与存储管理

- Dev
  — *定义服务的规模。*
  — *预测 UP/PBA（用户配置/业务活动模式）所定义的服务增长。*
  — *确定 DOD。*
- Ops
  — 确保数据存储能支持业务的增长。

- *记录作业描述信息并规范作业。*
- *确定DOD。*

- Ops
  - 软硬件全生命周期管理。
  - 日常运维操作。
  - 定义并管理作业进程、作业流。
  - 作业描述的定义与管理。
  - 作业异常处理。
  - 暂停（on hold）一个作业：
    - 当条件满足后，将作业退出暂停状态，该作业将被执行。
    - 依赖该作业的作业将不会被执行。
  - 搁置（on ice）一个作业[1]：
    - 在退出搁置状态后，该作业将被执行，但仅在下次运行时被执行。
    - 依赖此搁置作业的作业会被执行。

DevOps 与清理

- Dev
  - *定义哪些任务手工执行、哪些任务自动执行。*
  - *确定DOD。*
- Ops
  - 清理系统文件和应用程序。
  - 检查基础设施及应用程序的设置。
  - 建立健康性/完整性检查机制。
  - 检查系统和应用程序的行为。

DevOps 与性能调优

- Dev
  - *定义哪些信息需要被记录和调优。*

1 暂停（on ice）与搁置（on hold）的区别：on ice 的作业被暂停，但是依赖该作业的后续作业依然会被执行；on hold 状态的作业被暂停，依赖该作业的后续作业将不会被执行，直到这个作业恢复执行并成功执行。——译者注

- 监控——*应用程序监控。*
- 作业调度——*应用程序调度。*
- 清理——*应用程序清理和归档等。*
- 性能调优——*应用程序性能调优。*
- 存储管理——*数据库容量管理。*
- 打印管理——*应用程序相关的打印功能。*
- 环境管理。

从 Ops 的主要特征（监控、作业调度、清理、性能调优和存储管理），可以识别出更详细的分界线。下面给出了这些分界线。

DevOps 与监控

- Dev：
  - *为每种应用程序类型定义一个事态 SRG（标准、规则和指南）。*
  - *维护事态目录。*
  - *配置监控设备。*
  - *测试监控操作。*
  - *在 DOD 中记录事态目录 ID。*
  - *应用程序事态的定义和管理。*
- Ops
  - 软硬件生命周期管理。
  - 日常运维操作。
  - 黑名单/白名单定义与管理。
  - 基础设施/应用程序的事态日志文件分析。
  - 基础设施/应用程序的事态过滤。
  - 基础设施/应用程序的事态关联。
  - 基础设施/应用程序的报警。

DevOps 与调度

- Dev
  - *包含报表、流程、批处理、文件清理等的冲刺计划。*
  - *为计划定义一个 SRG。*

29.4 最佳实践

下面先描述 Ops 流程，然后阐述如何确定 Ops 流程与 Dev 流程的分界线。

29.4.1 Ops 流程

Ops 流程包括以下几个方面：

- 安装基础设施及应用程序。
- 配置基础设施及应用程序。
- 备份与恢复。
- 监控基础设施及应用程序。
- 作业调度（Job Scheduling）。
- 清理。
- 调优基础设施和应用程序的性能。
- 存储管理。
- 打印管理。
- 环境管理。

29.4.2 Ops 分界线

在过去的 10 年中，Ops 流程被极大地重塑了。首先，很多被管理对象都更加智能化；其次，云服务承担了很大一部分 Ops 工作；最后，“基础设施即代码”（Infrastructure as Code：IaC）将很多 Ops 工作转移给 Dev 团队。通过将大多数的物理对象虚拟化成可编程的软件对象，使 IaC 成为可能。

Ops 流程的分界线可以被形象地展示如下。正体字是与 Ops 相关的内容，斜体字是与 Dev 相关的内容。一般而言，Ops 用来满足整体的运维工作，Dev 则专注于有应用程序特性的 Ops 工作。

DevOps 的特征

- 安装——*在开发测试阶段的工具和应用程序。*
- 配置——*在开发测试阶段的工具和应用程序。*
- 备份与恢复。

29

任务划分（#28）

29.1 引言

Dev 和 Ops 需要协同工作。然而我们如何来划分任务的分界线呢？怎样才是一个好的划分方式，或者运维人员最终也都会成为开发人员？本文给出了一个解决方案。

29.2 术语

运维（Ops）流程

Ops 在很多框架中被认为是一个功能，这妨碍了为运维工作制定符合 SMART 原则[1]的清晰的目标，而且对功能进行治理也不够清晰。因此，本文认为 Ops 是一个流程。

29.3 概念

分界线

该术语表示将“左边”和“右边”进行分割，“左边”指 Dev，“右边”指 Ops。这种分割考虑了传统 Ops 团队执行的所有运维任务。

---

1 SMART 原则：①目标必须是具体的（Specific）；②目标必须是可以衡量的（Measurable）；③目标必须是可以达到的（Attainable）；④目标必须和其他目标具有相关性（Relevant）；⑤目标必须具有明确的截止期限（Time-based）。——译者注

— 集成：连接（ETL）。

- LCM 包括
  — 生命周期管理：发布、版本和补丁。

LCM 用于 DevOps 团队开发或配置的软件。LCM 适用于操作系统、数据库管理系统、软件包，如 Oracle OFSAA，SAP，Siebel 等。CEMLI 变更频繁，成为部署流水线的基本事务。LCM 变更影响重大，在大部分使用 DevOps 方法的公司中，变更仍然受到很大限制。注意，应用程序和工具都被当作配置项来对待，最后但同样重要的是，基础设施组件也被当成配置项。

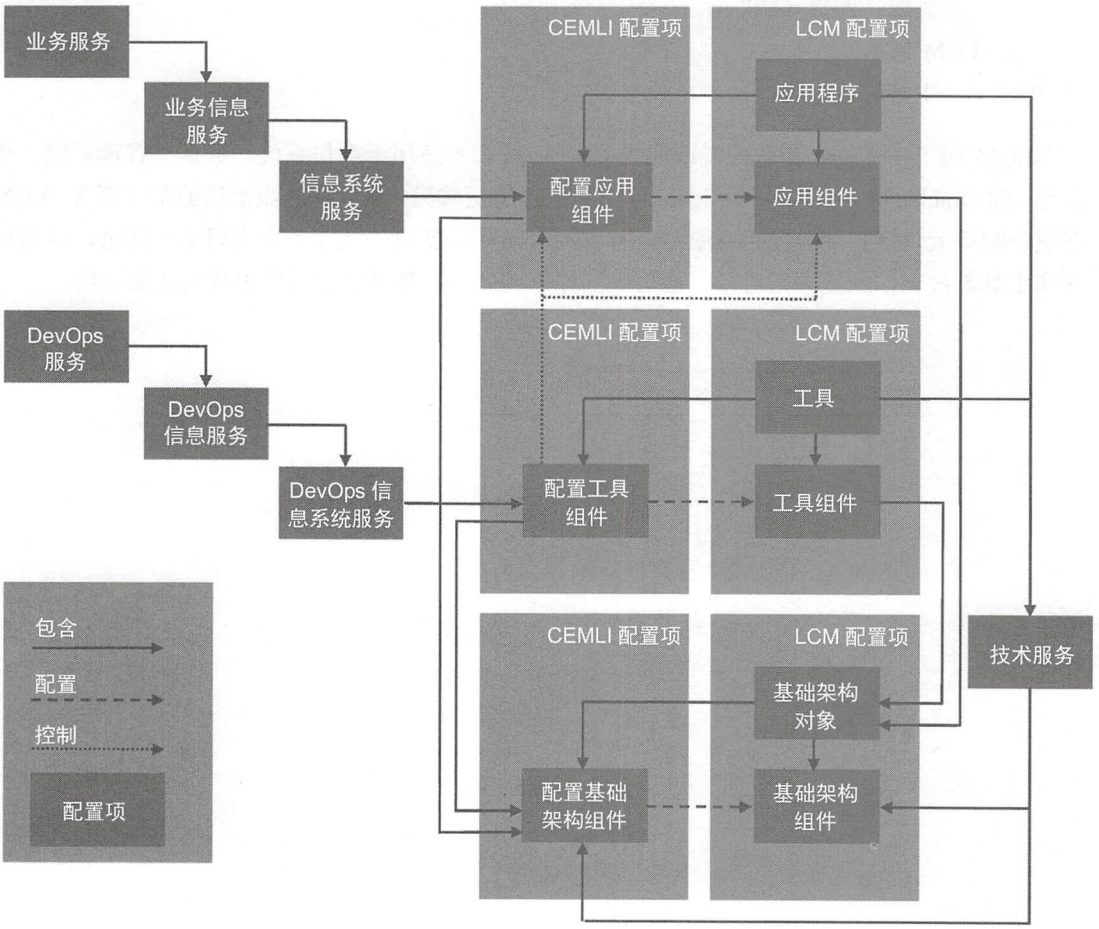


图 28-2 服务模型

28.4.2 两类产品

一类是 CEMLI，另一类是生命周期管理（LCM）。

- CEMLI 包括
  - 配置：业务规则等。
  - 扩展：额外的数据元素（数据模型）等。
  - 修改：代码调整（插件）。
  - 本地化：货币、日期格式等。

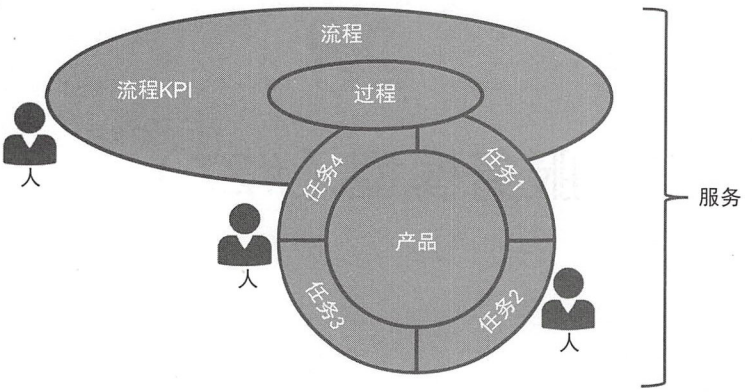


图 28-1 什么是服务

28.4 最佳实践

本文首先阐述可识别的两类服务：第一类是向客户提供的服务；第二类是 DevOps 团队内部使用的服务。这两类服务和更低级别的服务都定义为配置管理数据库（CMDB）中的配置项（CI）。接下来，本文将描述用于构建服务的产品类型。这些产品和它们的关系也定义在 CMDB 中。因此，在图 28-1 中所有的白色字体块，是 CMDB 的配置项。这样 CMDB 可支持风险和影响分析。越来越多的 CMDB 工具具备基于 CMDB 内容进行图形可视化的能力。

28.4.1 两类服务

从架构角度，可以识别出几种服务。有一种是图 28-2 描述的业务服务，这可能是一个客户服务或支付服务等。业务服务由业务信息服务支持。

这项服务是为了业务服务交付提供所需信息。CRM（客户关系管理）服务是一个例子，信息服务依赖作为服务提供的应用程序，这实际上是一个 SaaS（软件即服务）。一个例子是云上的 SAP。除了为业务提供的服务，还有为 DevOps 团队自身提供的服务。部署流水线可以看成一个信息系统服务，一套通过缩短产品上市周期来增值的工具。DevOps 团队在部署流水线上获取信息服务，比如在部署流水线中某处的对象状态。

28

服务模型（#27）

28.1 引言

ITIL 2011 是一种基于服务的方法。在 DevOps 的世界里，很多团队只专注于产品。本文将阐述以服务为导向如何在 DevOps 中应用。

28.2 术语

服务

确保以约定的价格所交付的服务真正为业务增值，并且确保所有已知风险都得到化解。

28.3 概念

什么是服务

服务不仅仅指一种产品，通过增加价值，使产品成为一种服务。例如，可以通过为打印机添加任务，如再次装入纸张、更换墨粉等，使之成为打印服务。仅仅向产品添加任务是不够的，任务必须及时得到执行。因此，需要一个服务级别协议（SLA）来约定提供服务的人员的表现。交付服务的流程经理对绩效进行监控。任务的处理速度取决于流程的关键绩效指标（KPI）。KPI 和 SLA 标准一致（见图 28-1）。

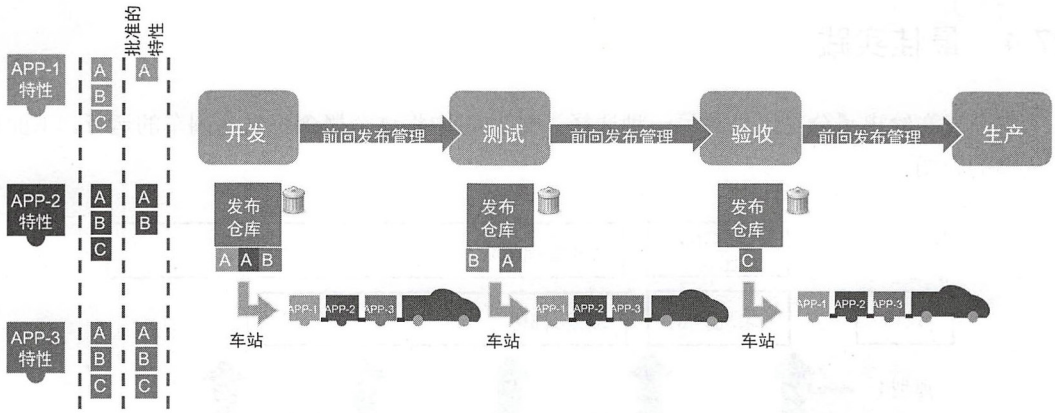


图 27-2 运行的列车

27.4.3 列车上有什么

发布管理团队决定一个包裹（用户故事）是否可以乘车。这个团队的成员由变更经理、发布经理和测试经理组成。决策的依据即车票，就是该用户故事是否通过了测试，是否拿到了发布授权。然而，发布管理团队必须判断用户故事之间是否存在冲突。没有上车的用户故事会滞留在车站里，待下一趟列车到来时，它们可能因测试通过而被批准上车，或者因存在缺陷而被再次拒绝上车。

27.4 最佳实践

前向发布给出了分支之外的另一种选择，图 27-1 中将这个概念比喻为列车的运行。下面进行具体的介绍。

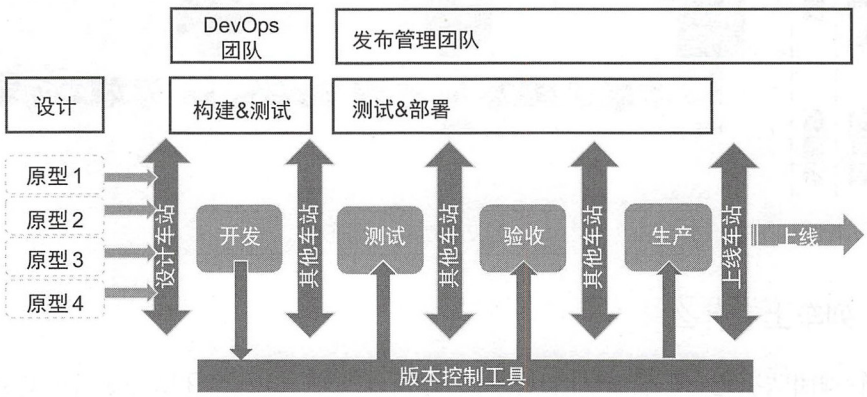


图 27-1 列车

27.4.1 列车

如图 27-1 所示，部署的对象犹如一个个被装载到发布专列上的包裹，每个包裹上车前应已出示了合法的车票，这个车票即完整的测试报告。“开发”“测试”“验收”“生产”作为铁路上的一系列火车站将对包裹进行验票以决定其是否可以运出。发布经理犹如列车工程师。然而，包裹实在太多了，发布经理无法决定能否发车，他必须在发布管理团队（Release Management Team, RMT）的协助下才能决定列车是否可以开出。

27.4.2 列车如何运行

图 27-2 中描绘了多辆列车，这是因为每个车站将检查不止一种车票。验票需要花费时间，所有的列车不是从“开发”直达“生产”的。任意时刻，所有的列车同时运行且仅从一个站点开往下一个站点。以一段时间来看，可能两周内有三趟列车将会运行：“开发”至“测试”、“测试”至“验收”，以及“验收”至“生产”。

27

前向发布（#26）

27.1 引言

文章#17 已提及了“前向发布”的术语，该发布管理模式的更多内容将继续在本文中讨论。

27.2 术语

软件配置项

软件配置项（S-CI）是 DevOps 开发过程所管理的对象的描述。为了描述一个对象或组件，软件配置项可以包含多样化的属性。第三或第四代语言的元数据、SQL 脚本、数据库结构及需要被某个应用程序所处理的任意对象均可作为软件配置项的属性。

27.3 概念

前向发布

前向发布的思想是，已部署到某一环境中的任意软件配置项，只要其发布没有出错，就将不再恢复。这是因为一条部署流水线常常被多个 DevOps 团队使用，一旦某个已部署的软件配置项被重置，所有依赖它的其他软件配置项也应被一并重置，虽然干扰可能很小，其余的测试用例运行良好。因此，应该等待随正常部署流水线而来的缺陷补丁。前向发布可防止分支。

步骤 5 持续优化部署流水线，从而消除浪费

有效时间和消耗时间如图 26-2 中所示，我们需要分析部署流水线中的延迟，如步骤 6 和步骤 7 之间有明显的等待和浪费。通过进一步分析，可以发现在 HP ALM 系统中的测试用例必须通过手工方式录入自动化测试工具 Fitnesse。这个步骤所消耗的 30 小时中，有 22 小时的浪费是可以通过进一步的自动化来优化的，如通过自动化的方式将 HP ALM 系统中的测试用例加载到 Fitnesse 中。

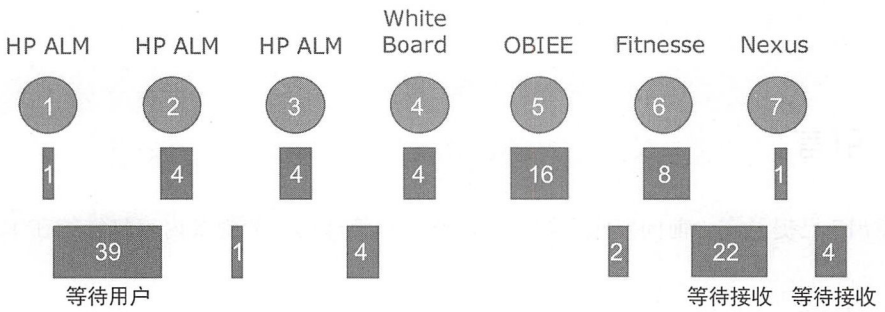


图 26-2 流程中有效时间和消耗时间分析

另一个瓶颈点在步骤 7，发布工作实际只花费了 1 小时，但是消耗了 5 小时。其中检查 Fitnesse 结果状态浪费了 4 小时，因为 HP AML 系统中的特性状态是由人工确认 Fitnesse 报表之后手工标记的。实际上这个步骤可以通过测试完成后将 Fitnesse 报表自动返回 HP ALM 系统来实现自动化。同样可以通过 Nexus 实现自动触发报表用于生产部署。

通过以上优化工作，部署流水线得到大幅优化，产品从构思到上市时间（Time To Market，TTM）缩短了 26 小时，这意味着节省了 23.6%的时长（26/110×100% = 23.6%）。

续表

步骤	输　入	过　程	输　出	工　具
7	测试输出结果	如果没有发现缺陷则发布报表	发布报表	Nexus（业界知名的制品管理工具。——译者注）

步骤 4　识别浪费

可以在表 26-2 和表 26-3 中增加两列来帮助我们识别浪费，即该步骤的有效时间和消耗时间。有效时间代表了创造价值的时间，而消耗时间包含着等待和浪费。所谓浪费是指没有给最终结果带来增值的时间。

表 26-3　流程消耗时间和有效时间

步骤	输　入	过　程	输　出	工　具	有效时间（小时）	消耗时间（小时）
1	用户故事	同利益相关方明确新报表的需求	已登记的特性	HP ALM	1	40
2	已登记的特性	完善冲刺待办事项列表	优先级已排的特性	HP ALM	4	5
3	优先级已排的特性	请利益相关方为每个优先级已的排特性定义测试用例	已登记的 Given-When-Then 测试用例	HP ALM	4	8
4	已登记的 Given-When-Then 测试用例	请开发团队定义每个特性的具体开发任务	填充后的冲刺看板	白板	4	4
5	填充后的冲刺看板	完成任务及报表	在 OBIEE 库中创建报表	OBIIE 库	16	18
6	在 OBIEE 库中创建报表	测试报表	测试输出结果	Fitnesse	8	30
7	测试输出结果	如果没有发现缺陷则发布报表	发布报表	Nexus	1	5
总计（小时）						110

表 26-1 详细描述了流程中的每个步骤。

表 26-1 流程步骤分解

步骤	输　入	过　程	输　出
1	用户故事	同利益方明确新报表的需求	已登记的特性
2	已登记的特性	完善冲刺待办事项列表	优先级已排的特性
3	已排优先级的特性	请利益方为该特性定义测试用例	已登记的 Given-When-Then 测试用例
4	已登记的 Given-When-Then 测试用例	请开发团队定义每个特性的具体开发任务	填充后的冲刺看板
5	填充后的冲刺看板	完成任务及报表	在 OBIEE 库中创建报表
6	在 OBIEE 库中创建的报表	测试报表	测试输出结果
7	测试输出结果	如果没有发现缺陷则发布报表	发布报表

步骤 3　定义部署流水线

在表 26-2 中列出了每个步骤相关的工具。

表 26-2 部署流水线

步骤	输　入	过　程	输　出	工　具
1	用户故事	同利益方明确新报表的需求	已登记的特性	HP ALM（HP 已应用生命周期管理软件。——译者注）
2	已登记的特性	完善冲刺待办事项列表	优先级已排的特性	HP ALM
3	优先级已排的特性	请利益方为每个优先级已排特性定义测试用例	已登记的 Given-When-Then 测试用例	HP ALM
4	已登记的 Given-When-Then 测试用例	请开发团队定义每个特性的具体开发任务	填充后的冲刺看板	白板
5	填充后的冲刺看板	完成任务及报表	在 OBIEE 库中创建报表	OBIIE 库
6	在 OBIEE 库中创建报表	测试报表	测试输出结果	Fitnesse（一个开源的协作验收测试框架。——译者注）

个项目中流水线尚未经过验证，但对于流水线的引入和实施，在团队内部已经达成共识。

26.4.1 步骤

创建一个部署流水线的步骤如下：

- 步骤 1 选择项目
- 步骤 2 识别工作流
- 步骤 3 定义部署流水线
- 步骤 4 识别浪费
- 步骤 5 持续优化部署流水线，从而消除浪费

步骤 1 选择项目

选择一个项目，进行部署流水线价值流映射。如果一个组织刚刚开始试点 DevOps，避免选择那些复杂的项目是比较明智的做法。大部分组织倾向于从低风险的项目开始实施 DevOps，例如，一个部署内容的改进项目。

步骤 2 识别工作流

识别从需求到部署的完整流程中的步骤：

- 有哪些输入、过程和输出？
- 谁负责执行该步骤？

图 26-1 给出了一个从提出报表需求到部署该报表的流程分析示例。

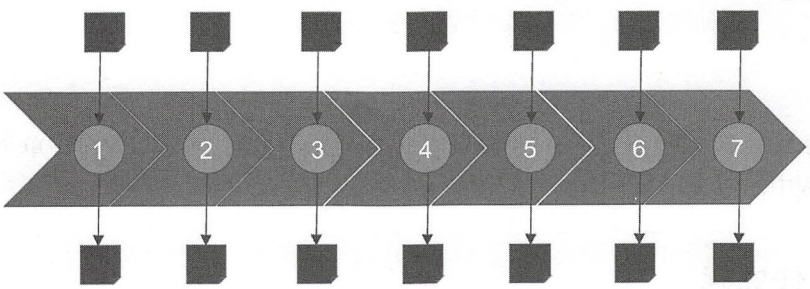


图 26-1 流程分析

26

部署流水线（#25）

26.1 引言

无论从任何角度来看 DevOps，部署流水线都是其中最具有吸引力的特性之一。

26.2 术语

部署流水线

部署流水线可以支持一个 DevOps 团队的完整交付过程，从提交阶段的源代码编写、签入，到生产环境中对象的实施结束。不仅如此，端到端部署流水线提供了更加强大的管控能力，从而带来更高的效率，以及更快的反馈速度。

26.3 概念

单件流

部署流水线最重要的特性之一在于确保整个 DevOps 团队成员可以聚焦于单个特性，从而减少整个流程中的浪费。

26.4 最佳实践

很难描述一个部署流水线应该是什么样子的，因为它对于每个组织来说都应该是独一无二的。本文介绍了一个简易方法来帮助你更好地理解流水线。这个方法源于一个既有项目，在这

逻辑测试用例示例

表 25-3 展示了表 25-2 中已识别风险的逻辑测试用例，这些逻辑测试用例依然要被转换为物理测试用例（意为真实测试用例。——译者注）。为了使这些真正的测试用例可以重复使用，需要提供变量。

表 25-3 逻辑测试用例

对象	类　型	风　险	测试用例
表	数据模型	基数错误	手动：检查实体类型之间的关系
表	数据模型	同义词	手动：校验企业数据模型里的实体类型及属性类型的唯一性
表	数据模型	同音词	手动：如果实体类型和属性类型已经以不同的名称存在，验证它们的名称
表	数据模型	无版本管理	自动：检查数据模型提取是否包含版本号
表	DDL 脚本	无版本	自动：检查 DDL 脚本及是否有版本号
表	DDL 脚本	数据库使用不正确	自动：检查 DDL 脚本是否以<database>开头
表	表格	不遵守命名规范	自动：检查所有对象的名称是否符合命名规范
表	表格	权限设置错误	自动：确认指定的角色是否有权限插入、更新、删除和筛选记录
表	属性	不允许有 null 值	自动：验证属性被正确地允许或无效
表	索引	非集群	自动：检测 DDL 语句是否产生了一个集群索引
表	索引	没有唯一索引	自动：输入两个相同键值的记录
表	触发器	不删除子记录	自动：删除记录并检查底层表格是否有子记录
表	视图	字段太少	自动：比较数据模型脚本与 DDL 脚本
表	DML	有“无异常”处理	自动：在完整性规则（integrity rule）内，产生一个错误提示，并检查错误消息结构

建议 4　创建测试用例

最后，完成测试模式和变量，测试系统生成测试用例，可能再补充另外的独立测试用例。

25.4.2　模式语言

表的模式可以用提取、转换、加载（Extract Transform Load，ETL）功能等进行扩展，基于模式总是以特定组合的形式被运用这一事实，我们可以识别模式语言，并对其加以应用。

建议 2 模式风险

分析已经过充分测试的对象有哪些部分已完成测试。也许在前面的步骤（模式识别）中已经有了一个好的开始，需要确定哪些风险可由测试用例控制。一般而言，这是上一步骤的扩展，主要检查交付的成果。这一步更为深入，因此需要专家的观点。如果下次遇到类似的测试对象，就能明确应及时控制哪些风险了。

模式风险示例（见表 25-2）

以表 25-1 的模式为例，须识别以下组件和其中的风险。例如，重复两次输入相同的唯一键，或者为非空（non-null）属性填充空（null）值，然后将每个组件的风险附加到表上。一个风险是否应该被测试，取决于风险发生的可能性和其产生的影响（可能性×影响）。因为最终会生成测试用例，所以从长期来看控制风险的努力将为零。

表 25-2 模式风险示例

对 象	类 型	风 险
表	数据模型	基数错误
表	数据模型	同义词
表	数据模型	同音异义词（Homonyms ）
表	数据模型	无版本管理
表	DDL 脚本	无版本
表	DDL 脚本	数据库使用不正确
表	表	不遵守命名规范
表	表	权限设置错误
表	属性	null 值不允许
表	索引	非集群
表	索引	没有唯一的索引
表	触发器	不删除子记录
表	视图	字段太少
表	DML	有“无异常”处理

建议 3 模式测试用例

为每个对象和每个风险定义一个测试框架。每个测试框架是一个测试用例，把它的值参数化，然后将所有的测试框架放在测试框架集中。下次需要测试对象时，在测试框架集中选择即可。

25.4 最佳实践

测试用例用于测试对象的功能和质量。对象可以是产品或服务，然而并不是单一的。例如，数据库中的表格就是一组对象：主键、外键、属性、索引和约束。如果要测试一个表格对象，就要考虑某些类型的测试用例。对于下一个要测试的表，可以重新考虑所有测试用例。最好假设表是基于模式进行测试的。本文对这种模式的思想进行了深化。

25.4.1 模式成熟度

学习使用模式需要一定的时间，但时间和金钱成本很快就可以收回。以下给出明确的模式化阶段：

- 模式识别。
- 模式风险。
- 模式测试用例。
- 创建测试用例。

建议 1 模式识别

为需要频繁测试并且包含许多测试用例的对象定义一个模式。定义模式的名称，创建一个检查表，以确保针对该对象的测试集是完整的。这样至少在下一次测试类似的对象时，有一个完整的检查表来测试完整性，如“错误！未找到引用源”。

创建模式表示例（见表 25-1）

表 25-1 创建模示表

对象	检查表
表	数据模型已调整并版本化
表	数据定义语言（DDL）脚本已创建并已版本化
表	表已创建
表	索引已创建
表	触发器已创建
表	视图已创建
表	基本的 DML 脚本已创建

25

测试模式（#24）

25.1 引言

在实际的项目中，创建测试用例会存在很大程度的浪费。如果比较这些测试用例，不难发现其中蕴含着的模式。因此，如果测试用例从一个模式中派生而来，将有助于提高测试用例的覆盖率。本文节将针对测试模式进行阐述。

25.2 术语

模式

模式是针对常见的和重复出现的问题提供的通用解决方案。通过分析 DevOps 团队所面临的问题及其解决问题的方式，可以创建模式，并且通过讲故事的方式与其他团队进行分享。本文将给出一些模式范例，供 DevOps 团队参考。

25.3 概念

模式语言

模式通常对应着相应的模式语言，这就意味着使用相同模式语言的模式在一起提供了一个通用的解决方案来解决更高层次的普遍问题。

- 冒烟测试（使用少量测试用例，看看是否能端到端连接）。
- 可用性测试（表明用户交互是否友好的测试用例）。
- 探索性测试（这种测试方法针对测试期间的演进，测试用例不是预先定义的，而是在测试运行期间创建的）。
- 演示（对用户进行新产品演示）。

24.4.7 生产测试阶段

在生产环境中至少要进行一次冒烟测试，然后才能发布。但是，在类生产环境中的测试用例也可以在生产环境中执行。在以前手工配置很普遍的情况下，配置检查会被认为是 PAT 的一部分。现代的环境管理避免了这些错误，然而冒烟测试仍然很有价值。

24.4.2 部署流水线

表 24-1 显示了各个测试类型在通过部署流水线时分布的情况。

表 24-1 部署流水线各阶段的测试类型

提交阶段	验收测试阶段	容量测试阶段	手工测试阶段	生产测试阶段
UT	CT	PST	UAT	PAT
Pre-CT	IT			
	ST			
	FAT			
	NFR			
	Pre-PST			
	Pre UAT			

24.4.3 提交阶段

提交阶段的速度必须非常快。在正常情况下，本地自动构建的标准是 5 分钟。对单元测试用例来说，不允许它连接到基础设施服务的其他应用程序中。在通常情况下，它只是一个或几个测试用例，使用测试驱动和/或测试桩。

在提交阶段会执行一些组件测试用例来检查组件之间交互的正确性，这样可以在主干分支上签入代码的时候减少对构建工作的破坏。

24.4.4 验收测试阶段

验收测试阶段用于执行较长时间的测试用例。

24.4.5 容量测试阶段

一个 PST 可能需要花费一天甚至更多的时间来运行。在验收环境中运行此测试将破坏部署流水线，因此，对于 PST 需要有一个单独的环境。

24.4.6 手工测试阶段

持续部署意味着没有手工测试。在不需要用户验收的时候，这是可能的。在持续部署的情况下，仍有手工测试，也可以选择进或不进流水线。这个阶段可以执行的测试用例是：

24.3 概念

主测试计划

主测试计划用于制定测试策略。

测试策略的制定，是为了确定项目需要的测试类型，以及在何时、何地、由谁来执行，谋求以最小的努力根据需求测试最重要的功能。

24.4 最佳实践

由于几十年来测试管理的最佳实践都是在 TMap NEXT，BiSL，ASL 和 ITIL 中定义的，因此，本文将重点讨论部署流水线和测试管理最佳实践之间的关系。部署流水线需要尽可能地自动化测试用例。为了能够在部署流水线上清晰地描述测试类型，它们应该更加明确。本文是基于测试类型的洋葱环展开的。下面，本文将描述各个测试类型在部署流水线中的位置。

24.4.1 测试类型洋葱环

图 24-1 是用洋葱环的形式对测试类型进行相互关联的，这些不同的层用来表示执行测试用例的顺序。

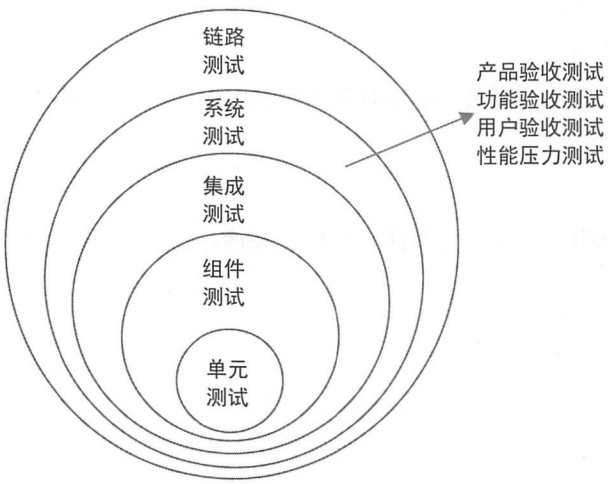


图 24-1 测试类型洋葱环

性能压力测试

性能压力测试用于验证在用户数量、事务类型及背景工作负载已知的情况下，系统是否有能力及时处理请求。

产品验收测试

产品验收测试用于验证应用系统是否在目标环境下正常工作。

系统测试

系统测试用于检查整个应用系统是否满足已定义好的需求。

测试驱动程序

测试驱动程序是一个虚拟的软件（dummy piece of software），用于调用独立的函数、组件或完整的应用程序。通过这种方式，测试用例可以调用测试驱动程序，从而执行单元测试用例、组件测试用例、集成测试用例或系统测试用例。

测试桩

测试桩是用于模拟功能、组件或整个应用程序之间交互的虚拟软件。

单元测试

一个单元测试只负责测试一个小对象的测试用例，如一个 S-CI。

用户验收测试

用户验收测试用于确定在可用性、用户友好性、信息注册难易程度等方面，应用系统是否能够满足用户。

24

测试类型（#23）

24.1 引言

有许多种不同的测试，问题是我们在何种情况下执行何种测试，本文将会提供一些建议。

24.2 术语

链路测试

链路测试用于保证在同一个应用链路上的应用程序能够正常工作。有时候这种测试又被认为与集成测试一样，但是建议把它命名为链路测试，与应用模块间的集成测试区分开来。

组件测试

大型应用系统构建于组件与模块之上。每个组件都有几个 S-CI 组成，组件测试用于测试这些组件。

功能验收测试

功能验收测试用于验证应用系统的主要功能是否满足需求。另外，非关键功能错误及异常处理功能也需要测试。

集成测试

集成测试验证系统的组件/模块是否能组合在一起正常工作。它与组件测试的功能有一些重叠。尽管如此，这种测试聚焦于多个组件间的测试，而组件测试一般只聚焦于一个组件。

续表

要 求		
R#	要 求	链 接
T3	测试用例必须与 CI 关联	这需要链接到 CMDB 工具
T4	测试执行	如果 Given（先决条件）—When（特定条件）—Then（满足特定条件的下一步）是在工具需求中定义的，那么需要链接此工具
T5	在运行中一组测试用例的执行是被管理的	
T6	测试运行必须参照已定义的测试用例	如果测试运行和测试用例没有在同一个工具中定义，则需要另外的链接
缺陷管理工具		
D1	缺陷是唯一标识的	
D2	缺陷和测试运行 ID/ 测试用例 ID 之间有一定的关系	这需要链接到测试自动化工具
文件生成工具		
G1	文档是基于一个独特的 ID 注册的	
G2	文档是基于基线的	这需要链接到源代码库工具
G3	文档是生成文档的构建	这需要链接到构建服务工具

表 23-7 文档元数据

缺陷				
F#	类型	名称	类型	描述
1	PK	DocID	Integer	文档的 ID
2	FK	Baseline	Integer	有效文档的基线 ID
3	FK	BuildID	Integer	生成文件的构建 ID
		…	…	…

23.4.3 工具要求

根据元数据表，表 23-8 中定义了许多工具要求。

表 23-8 工具要求

要求		
R#	要求	链接
源代码库工具		
S-01	每个源对象都有一个独特的 ID	
S-02	基线的签入是可自定义的	可能是一个可跳转到活动目录工具的链接
S-03	源代码库必须能够定义依赖关系	
S-04	源代码库必须能够将 S-CI 关联到 CI	这需要链接到 CMDB 工具
S-05	编译器必须能够更新 S-CI，登记最后的构建 ID	这需要链接到构建服务工具
S-06	源代码库必须使用版本控制	
构建服务器工具		
B-01	构建服务器必须登记已触发的构建	
B-02	构建服务器必须管理构建的基线 ID	这需要链接到源代码库工具
B-03	构建服务器必须查找依赖关系或能够根据源代码文件来确定它们	这需要链接到源代码库工具
B-04	构建服务器必须将对象 ID 关联到 CI	这需要链接到 CMDB 工具
B-05	构建服务器必须管理对象放置在对象制品库中的位置	这需要链接到产品库工具
自动化测试工具		
T1	测试用例必须是唯一注册的	
T2	测试用例必须与对象 ID 有关系	这需要链接到产品库工具

续表

测试用例				
F#	类型	名　称	类　型	描　述
7		Post-condition		一旦用例执行失败，需要知道当时执行的结果，包括数据库中的数据、屏幕上显示的信息等
8		ExpectedResult		测试成功或失败
		…	…	…

表 23-5　测试运行元数据

测试运行				
F#	类型	名　称	类　型	描　述
1	PK	TestRunId	Integer	测试运行的 ID
2	FK	TestCaseId	Integer	在测试运行中的测试用例 ID 对于作为测试运行一部分的测试用例，必须加以记录
3		DateTime	DateTime	如果填充此字段，则测试运行必须等待触发器启动的时间
4		Result	Varchar(255)	测试的结果
5		ContinueFlag	Char(1)	若测试用例失败，Y 代表继续执行，N 代表停止执行。对于单元测试用例，该标志始终设置为 Y
		…	…	…

表 23-6　缺陷元数据

缺　陷				
F#	类型	名　称	类　型	描　述
1	PK	DefectID	Integer	缺陷的 ID
2	FK	TestRunID	Integer	进行测试运行的 ID
3		TestCaseID	Integer	导致缺陷的测试用例的 ID
		…	…	…

续表

构建				
F#	类型	名称	类型	描述
4	FK	DependencyID	Integer	在编译此 S-CI 之前必须编译的 S-CI 的 ID。假定有一个依赖项，也可能有更多的依赖项，需满足 1：*N* 关系（附加表）
5	FK	CI-ID	Integer	构建所属的 CMDB 的 ID
6		BuildDateTimeStart	DateTime	构建开始的日期和时间
7		BuildDateTimeEnd	DateTime	构建结束的日期和时间
8		BuildDuration	Integer	构建持续时间的秒数
9		CompileErrorNumber	Integer	构建中出现的错误数量
10		CompileWarningNumber	Integer	构建中出现的警告数量
11		…	…	…

表 23-3 对象代码元数据

对象代码				
F#	类型	名称	类型	描述
1	PK	ObjectCodeID	Integer	对象的名称
2	FK	BuildID	Integer	创建此对象的构建的 ID
3	FK	CI-ID	Integer	该对象所属的 CMDB 的 ID
4		…	…	…

表 23-4 测试用例元数据

测试用例				
F#	类型	名称	类型	描述
1	PK	TestCaseID	Integer	测试用例的 ID
2	FK	ObjectCodeID	Integer	被测试对象的 ID。如果有更多对象，则必须扩展数据模型以支持 1：*N* 关系（附加表）
3	FK	CI-ID	Integer	该对象所属的 CMDB 的 ID
4		TestType	Char(3)	UT, ST, SIT, FAT, PAT, UAT
5		Pre-condition		执行测试需要什么，如测试数据和授权
6		Test execution		Given（先决条件）—When（特定条件）—Then（满足特定条件的下一步）

23.4.2 数据模型

有针对不同类型的对象的构建，通过使用工具将这些对象相互关联起来。

为了识别对象的相互依赖关系，最好的方法就是使用数据模型。在数据模型里面，我们定义出对象、对象间的关系及对象的数据属性，这些可以用图形的方式来展现，用表格的方式展现也行。很重要的一点是，在需求确定了之后，要确定工具在多大程度上能覆盖所需的信息。通常，除了整理出一些表，还可以从配置中得到一些缺失的信息。接下来，如表 23-1 至表 23-7 所示，我们以一个数据模型作为示例，其中，PK 表示主键，FK 表示外键。

表 23-1 源代码元数据

源代码				
F#	类型	名　　称	类　　型	描　　述
1	PK	SourceCodeID	Integer	S-CI（软件配置项）的 ID
2	FK	BaselineID	Integer	S-CI 所属基线的名称
3	FK	DeveloperID	Integer	最后一次变更 S-CI 的开发人员 ID
4	FK	DependencyID	Integer	在编译此 S-CI 之前必须编译的 S-CI 的 ID。假定有一个依赖项，也可能有更多的依赖项，依赖项需满足 1：*N* 的关系（附加表）
5	FK	CI-ID	Integer	此 S-CI 所属的 CMDB 的 ID
6	FK	LastBuildID	Integer	使用 SID 的最后构建
7		Name	Varchar(255)	S-CI 的名称
8		Version	SmallInt	S-CI 的版本
9		DateTimeCheck-in	Datetime	最后一次代码合入的日期和时间
10		Change	Varchar(255)	代码变更的描述信息
11		…	…	…

表 23-2 构建元数据

构　　建				
F#	类型	名　　称	类　　型	描　　述
1	PK	BuildID	Integer	构建的 ID
2	FK	BaselineID	Integer	S-CI 所属基线的名称
3	FK	DeveloperID	Integer	触发构建的开发人员姓名

23.3 概念

工具集成

DevOps 使用自动化的部署流水线，使用的工具要么是集成的，要么是紧密耦合的。假如我们要使用更多的工具，集成是一个非常重要的方面。包括：

- 不同工具的术语保持一致性。
- 不同工具之间信息格式的交互。
- 流水线中交付的同步。
- 对工具进行状态跟踪。
- 基于更多工具库进行报告。

23.4 最佳实践

在文章#05 中提到，集成工具的架构选型，可以选择自建/采购不同的集成工具，或者选择每个种类的最佳工具。在文章#16 中描述了 CMDB 和 S-CMDB 之间的关系。

CMDB 和 S-CMDB 之间的接口不是唯一的（工具集成方面的）挑战，它贯穿 DevOps 整个部署流水线。我们看到这一问题不时被提出来讨论。本文介绍构建中集成工具的主要陷阱。

23.4.1 适用范围

构建必须自动化进行，并且是可重复的、稳定的和快速的。文章#21 介绍了构建是如何在 CI 中工作的。这里涉及以下对象：

- 源代码（源代码库）
- 构建（构建记录）
- 目标代码（制品库）
- 测试用例（测试数据库）
- 测试运行（测试数据库）
- 缺陷（缺陷数据库）
- 文档（源代码和制品库）

23

工具（#22）

23.1 引言

构建服务器需要协作的工具。本文介绍关于工具集成的需求。

23.2 术语

元数据

元数据是描述对象的信息。以构建为例，它的元数据可能包括：

- 构建 ID
- 构建时间
- 基线 ID
- 应用 ID

主键/外键

对象的 ID 是对象唯一的标识符。许多对象，如基线和构建都有关联。通过共享关键字段可以在这些对象之间定义关系。例如，在构建对象中，通过使用基线 ID，表明唯一的构建 ID 是基于哪个基线 ID 的，而基线 ID 是作为外键（FK）在构建对象中注册的主键。

续表

级别	级别定义	级别特征
3	优化级	• 每月审核各个团队的 DevOps 流程，确保满足持续集成的需求 • 制订改进计划以调整需求之间的偏差 • 可视化 DevOps 团队的偏差 • 成熟团队通过最佳实践交流来帮助不成熟团队提高水平

22.4.5 普遍使用的工具

持续集成工具有很多，例如：

- 版本控制工具
  - Concurrent Versions System (CVS)
  - GIT
  - Mercurial (Selenic)
  - RabbitCVS
  - Serena Dimensions
  - Subversion (SVN)
- 构建工具
  - Jenkins
  - Make (MK)
  - PVCS make
- 存储库工具
  - Sonatype Nexus
  - Artifactory

续表

级别	级别定义	级别特征
0	可重复级	必须进行以下改进才能达到 1 级： • 每个开发人员在本地具有一个构建服务器 • 在本地代码签入之前，要确保先将主干代码复制到本地并构建成功，不能破坏主干代码 • 确保本地合并代码没有任何错误时才能签入主干，从而避免开发人员相互影响 • 所有开发人员应认同在主干受到损害时协助解决问题，在主干恢复之前不能进行签入 • 在本地和主干级别使用 Commit 命令执行自动化构建 • 按照正确的顺序编译软件模块，确保在服务器端构建过程考虑了软件模块间的依赖关系 • 构建服务器标准化，所有开发人员使用相同的工具和脚本，基于一个统一的源文件库和制品库，使用统一的部署流水线 • 不仅要执行系统测试和系统集成测试，还要执行验收测试（功能和非功能） • 通过源代码库和/或源代码直接生成文档
1	一致级	必须进行以下改进才能达到 2 级： • 定义构建度量数据，例如： — 构建时间 — 发现的缺陷 — 质量标签 Tiobe（参见文章 #20） • 使用信息雷达展现不同的 DevOps 团队及其成员的绩效 • 定义度量目标，并告知员工实现这些目标所需要做的工作 • 监控度量数据的状态，以达成度量目标 • 避免出现在预定时间内未能修复构建的情况
2	量化管理级	必须进行以下改进才能达到 3 级： • 针对集成时出现的问题，定期召开会议进行原因分析 • 规划产品待办事项列表的度量数据，对工具做出调整 • 定期召开会议制定更有效的度量数据，从而更快地发现集成过程出现中断的情况 • 使用信息雷达展现度量数据，并据此采取措施
3	优化级	维持在 3 级必须采取以下步骤： • 定义满足持续集成所需的一系列需求

- 保持快速构建。
- 在类生产环境（克隆的生产环境）中进行测试。
- 确保易于取得最新构建的可交付成果。
- 每个人都能看到最新一次的构建结果。
- 自动化部署。

22.4.3 成熟度（见表 22-1）

表 22-1 持续集成成熟度 （www.devopsguys.com）

级别	级别定义	级别特征	构建管理和持续集成
3	优化级	关注过程改进	团队经常面对面讨论集成问题，解决问题应用自动化的工具，反馈速度更快，信息更透明
2	量化管理级	过程度量和控制	收集构建指标，使其可视化，有措施确保构建不会被破坏
1	一致级	在整个应用生命周期中应用自动化流程	每次变更提交时都会执行自动化构建和测试，可管理依赖关系，有可复用的脚本和工具
0	可重复级	已文档化流程并且部分过程实现自动化	可定期自动化地构建和测试，使用自动化流程可以对源代码反复进行任意构建
–1	退化级	不可重复的流程，难以控制并采取措施	人工的软件构建过程，没有制品管理和报告

22.4.4 发展路线（见表 22-2）

表 22-2 持续集成发展路线

级别	级别定义	级别特征
–1	退化级	必须进行以下改进才能达到 0 级： • 在一个源代码库中集中管理源代码 • 每天至少进行一次代码签入，大大减少合并所需时间 • 实现自动化构建，以便可以在没有手工操作的情况下完成（包括选择源文件、编译源代码、目标代码保存到指定的制品目录下） • 确保构建的自动化，对库代码进行自动化构建，通过标签提取库代码，实现可重复的构建 • 确保在构建时执行单元测试用例

- 持续集成的生命周期。
- 最佳实践。
- 成熟度。
- 发展路线。
- 普遍使用的工具。

22.4.1 持续集成的生命周期

图 22-1 展示了持续集成的概貌。CI 的生命周期从代码存储库（也称作 S-CMDB）中的源代码开始。通常源代码的代码存储工具包含版本管理功能，可以对需要构建的源代码标注发布标签，然后根据标签导出，在构建服务器完成编译工作。当目标代码生成后，就可以用已创建的单元测试用例来测试软件，并从源代码库和/或源代码直接生成文档，将目标代码存储在数据操作语言（DML）中。

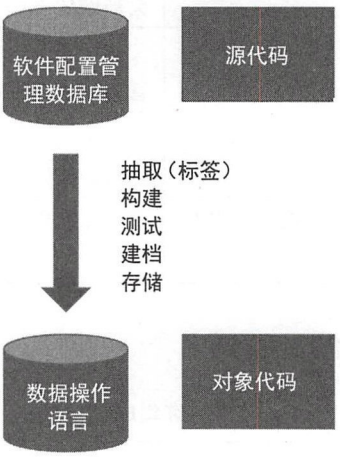


图 22-1 持续集成的生命周期

22.4.2 最佳实践

- 维护一个代码库。
- 自动化构建。
- 进行构建自检（self-testing）。
- 每人每天都要向基线提交代码。
- 每次提交（两个基线）必须进行一次构建。

作者简介

Bart de Best

拥有30多年的IT工作经验，目前担任荷兰飞利浦医疗数字病理解决方案产品的质量架构师、ING银行DevOps基础服务架构总设计师。曾在荷兰银行(ABN Amro)、 DHL、 荷兰皇家航空(KLM)、 联合利华、荷兰税务总局、英美烟草等各类世界知名企业担任IT组织要职。

自1999年至今，一直是EXIN全球专家库核心成员，不仅参与了早期EXIN的ITIL Manager出题，近年来还参与了基于欧盟e-CF的ICT人员能力测评系统和DevOps Master的考题与实践作业的设计工作。

2016年和2017年，Bart de Best先生作为EXIN DevOps Master资深导师，两次来到北京为中国培养了两批DevOps Master授权讲师。

随着新一轮数字化转型的深入，如何快速有效地交付数字化产品和服务成了各个组织日益重要的竞争能力。DevOps作为一种实现敏捷开发和运营的最佳实践，在某种程度上已成为数字化转型的标配。本书从理念、方法和工具等不同层面进行了系统介绍，是一本不可多得的DevOps宝典！

——CIO时代学院院长　姚乐

作者将其多年的IT管理经验与DevOps最佳实践进行了巧妙结合。作为EXIN的 DevOps认证的参考书籍，本书将对传统行业的IT从业者的知识迭代和能力提升起到积极的作用。在企业数字化人力资源培养方面，DevOps最佳实践的应用，助力企业改善内部协作，增强动力和创新力，提升企业外部竞争力。时下各企业在数字化进程中，对具有DevOps实践经验的综合性IT人才求贤若渴，EXIN作为全球IT管理著名机构，其所颁发的DevOps国际认证为企业DevOps人才的选拔提供了重要的参考依据和衡量标准。

——索迪斯（中国）人力资源业务合作伙伴　龙玥

DevOps作为一个数字化领域越来越重要的实践方法，已经得到业界的广泛认可。万达集团在下一步数字化转型中希望借鉴本书中的理论知识和实践方法。

——万达集团副总裁兼CIO　朱战备

本书从服务管理的视角阐述DevOps，和常见的软件开发或运维视角有很多不同之处。作为本书的主审，我非常敬佩作者的知识广度和深度。书中实际可操作的知识点很多，是值得DevOps学习者和企业相关负责人一看再看、反复研读的佳作。

——DevOps Master授权讲师、中国DevOps Days组织者 许峰

跟踪新书好书
投稿互动反馈
联系我们：sjb@phei.com.cn
(010)88254199
北京世纪波文化发展有限公司
http://www.century-vision.com
责任编辑：刘露明

ISBN 978-7-121-35116-7
9 787121 351167 >
定价：78.00元